A Robust Method for Visual Pose Tracking with LEDs

Ruben Stein
(Matriculation Number: 1955806)

Faculty 3 - Computer Science
University of Bremen
Germany

Type:	Diploma Thesis
Degree:	Dipl. Inf.
Date:	February 17, 2010
Supervisor:	MSEE Mark Edgington
1st Referee:	Prof. Dr.-Ing. Udo Frese
2nd Referee:	Prof. Dr. Kerstin Schill

Abstract

Pose tracking belongs to the classic tasks in robotic environments. There are many well-investigated methods which are based upon photogrammetric calculations. Nevertheless, the real-time extraction a robot's pose from images under changing lighting conditions remains a challenging task. To decide on a specific tool chain which transforms pixel values to spatial information involves balancing complexity, speed, and robustness.

This thesis focuses on the design, implementation and testing of a tracking method which is robust to changes in lighting conditions, and provides a clean separation between signal and background information. Furthermore, the presented method requires only minimal configuration for use in unknown environmental conditions.

In order to achieve this, used flashing Light Emitting Diodes (LEDs) were used as markers arranged in grids. With this setup, the feasibility of uniquely identifying different flashing LEDs using a sequence of monocular images is investigated.

Contents

List of Symbols

Symbols

Symbol	Name	Description
ι	Iota	Intensity value of a grayscale (shades of gray) image
$\Delta\,\iota$	Delta Iota	Intensity change between Frame F_t and F_{t-1}
j	Square-Root of -1	For the Fourier transform j represents $\sqrt{-1}$ in the context of complex numbers
X_k	Bin[1] Value	Frequency-Domain value X of the k^{th} spectral bin
x_t	Signal Value	time-domain value x at time t

Units

Symbol	Name	Description
fps	Frames per Second	A measure for the speed of image processing devices or software
Hz	Hertz	Recurrence of a periodic event per second, commonly used to classify frequencies
s	Second	Time unit

[1]See Glossary for a definition

1 Introduction

1.1 Motivation

Starting from a raw image a robot has no or little environmental knowledge, which can aid it in interpreting what he sees. Therefore, in machine vision knowledge about the environment must be derived from how raw pixel information is interpreted. Basically human vision faces the same problems, but there is a basic difference in terms of *context knowledge*. When a human sees a picture of an object, it is clear to him that this object represents a certain entity in his perception of the world. We look through a window and see a green meadow. The green is bright when the sun shines through a hole in the clouds' cover and changes to a grayish color as larger clouds pass by. Nonetheless we know that this is the same meadow we saw before with a different color, and that it is growing on the ground. It is therefore intuitive to us that a person walking on this meadow will occlude some parts of ground. Likewise an orange ball on the ground behind the person will remain where it is, and reappear after the person has walked past it.

What seems so obvious to us is in fact a collection of complex context, which is linked to the colors, textures and shapes we see. What was really observed by us were the reflections of light as waves of photons by some physical compounds in the visible spectrum of electromagnetic radiation. As our eye is able to categorize the frequencies of light waves into the visual bands of certain colors, we perceive a pattern of color information. This information is now grouped by some constraint, so for example equally colored areas will be recognized as connected. Subsequently the alignment of objects we have recognized tells us something about our own attitude towards the scene and so on. It falls into place that the perception of objects is an incremental process which may include arbitrary compositions of low- and high-level information. The latter can be formed from small bits of, considered alone less useful, information from low level entities, which is processed regarding certain attributes. This applies to human as well as digital image processing.

In machine vision the above mentioned human intuition about environmental context is not available. The algorithms can only use the limited information their sensors provide, all the subsequent reasoning has to be modeled using raw data. Such models may include the recognition of patterns, properties, temporal occurrence and so on. In [Gonzalez and Woods, 2006] digital image processing is described as a process taking place in three levels. The lowest level processes images and also returns images. Mid-level processes will, for instance, segment certain parts of the image to form intermediate knowledge. Finally the higher-level methods of image processing make *sense* of an image. An example of this would be how things are arranged on a table, or where in a scene an object is moving.

One well known way of interpreting image data is to register colored markers. For instance, in the RoboCup[2], a robotic soccer challenge, colored patterns are recognized and give information about a robot's location and velocity. Those markers can be found and identified on a single frame. The more general problem of *image segmentation* in terms of color is called *color slicing*.

[2]See [Röfer, 2008] for details

It involves thresholding the parameters of a color space in a way that a specific region of an image gets selected. Common color spaces are the Red-Green-Blue (RGB) space basing on the combination of primary colors or the more intuitive Hue-Saturation-Intensity (HSI) space. The latter was inspired by the human intuition in describing colors. In Figure 1.1 a specific region for *blue* is selected by thresholding the HSI color space.

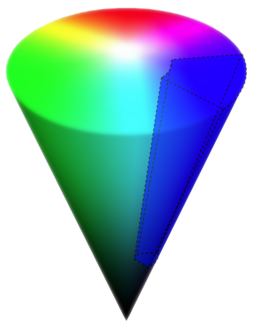


Figure 1.1: Selection of a *blue* range in the HSI color space

While this works well if the lighting is stable, problems arise in volatile environments including even sunlight. Objects will change their color shade based on the intensity they are illuminated with. Small changes in lighting can cause the “blue” of an object to leave the region covered by the slice in Figure 1.1. As we have only little influence to natural sunlight, algorithms to detect certain color features are sensitive to those changes. Therefore this thesis describes an alternative approach without leaving optical methods.

1.2 Problem Description

To overcome the problems of lighting dependent color-mappings, this thesis proposes a method for lighting-independent optical pose tracking in 3D. Its application area is real-time robotic navigation. In contrast to many other approaches, the focus of this method is to avoid the need of color codes for marker positions. This goal is accomplished by the use of frequency information gained from blinking Light Emitting Diodes (LEDs) instead of color segmentation. In doing so, the camera image is analyzed to find certain intensity changes over time. One challenge related to this task is the fact that a transformation of intensity information into blinking patterns must be done in real-time for a robot to navigate. For an stationary object, this is done by watching at the information of a pixel over time. In contrary to other approaches though, the used markers are not persistent in each frame, like a color feature would be. The latter provide absolute information about the position of an object enclosed in any frame, whereas the frequency is only interpretable after a sequence of images of the same scene. Hence for spread sequential image data, recognizing movement is more complicated than it is for persistent markers.

To summarize, the core of this work is the development of a method to use blinking LEDs as markers for position determination. A key advantage of this method is that it functions well in outdoor environments and is immune to changes in lighting conditions. Furthermore, the general

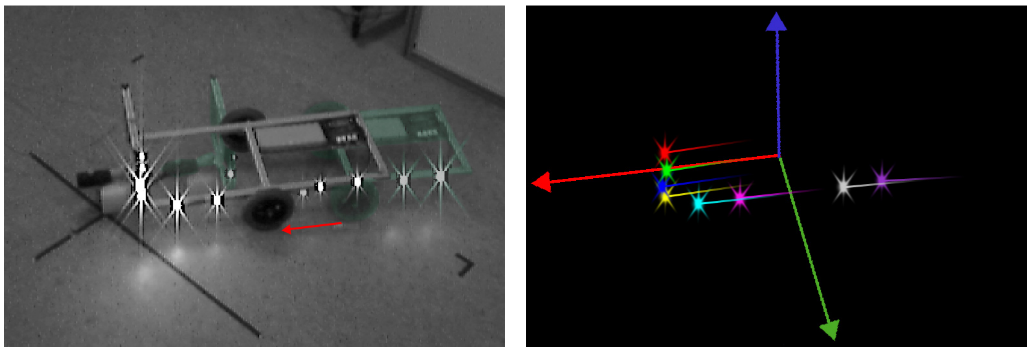

(a) Input (b) Output

Figure 1.2: As input the system gehts raw images from the camera. In (a) the older pose is drawn shaded to symbolize that the cart has already moved. Its movement is indicated by the red arrow. Out of this the system calculates the LEDs' positions and the attitude of the cart shown in (b). The lines in that image denote the trajectory of the LEDs.

focus is to avoid configuration effort where possible to achieve a robust on demand positioning without the need to calibrate to the current lighting conditions.

2 Scientific Placement

In general this thesis is situated in the field of machine vision, thus the whole approach incorporates general image processing techniques like BLOB building or region finding. Further, the proposed method is used as the recognition component of a robotic application capable of estimating a certain object's pose in space. The tools necessary for this kind of calculation are well known and can be found as photogrammetric algorithms [Bradski and Kaehler, 2008]. Although many problems are already solved, there are still areas of intense research. The so called *perspective n-point problem* is solved in many different ways, but there are still publications like [Xu et al., 2008], which explore new techniques. The problem describes how a machine can extract perspective information out of pictures. The motivation behind such intense research in robotic perception is, among other things, the fact that many problems in machine vision are no problem for a human observer with immense contextual information in mind.

The presented approach encloses the identification of unique binary patterns, hence it will touch digital signal processing. With the Fourier transform [Smith, 1997, Ch. 8], a classical tool for extraction of information from wave forms is used. To increase the performance, the special version named Sliding Discrete Fourier Transform (S-DFT) is utilized.

Apart from the Fourier transform as aid to identify certain blinking patterns, I analyzed the use of Hidden Markov Models (HMMs) [Russell and Norvig, 2003, Ch. 15.3]. Coming from the area of knowledge representation in artificial intelligence, this method is most common in speech and gesture recognition. Here it helps to distinguish between several aberrations of a signal such as a spoken word, which sounds different depending on the speaker.

To estimate the motion of objects, a Bayesian state estimator is used. Specifically a particle-filter [Thrun et al., 2005, Ch. 4.3] is used to predict the motion performed by a supervised object. As soon as a motion prediction is available, the pixels at the new object position can be used as further input for the frequency analysis taking place beforehand at the static, initial position. Based on this the hypotheses of the estimator can be checked for consistency with the collected frequency data, allowing a subsequent pose prediction.

Finally, methods of runtime optimization were used to make the whole approach suitable for real-time use. This involves both conceptual considerations like choosing fast algorithms as well as optimizing their implementation on the software level. Their importance for the system as a whole is crucial as only by this acceleration the approach is usable as intended. For real-time performance it is necessary to finish all calculations for a frame before the next frame arrives. For example, at an operational capturing rate of 30 fps (frames per second), this means all computation triggered by this frame has to be completed within 33 ms to be ready for the next frame.

2.1 Contribution

The main feature of this work is to provide a method for real-time robotic vision, which is independent of the lighting conditions of the surrounding environment. It can therefore be used outdoors and in real world applications, where artificial lighting constraints are difficult to ensure.

The idea is not new — from a cursory glance it appears to be nothing more than motion capturing with active markers. However, motion capturing techniques commonly rely on constantly illuminated features allowing BLOBs in the infrared light spectrum to be easily identified. The desired lighting independence is therefore not addressed by such techniques.

From the problem presented in Section 1.2, several subproblems can be identified. First, a method for the registration of pixelwise blinking patterns has to be developed. That was done by providing a very fast method for solving large scale Fourier analysis of continuous, discrete data. In a secondary step, a relative pose of the camera must be extracted from the data segmented by the foregoing frequency analysis. Afterward, the method must be transferred to a moving target in order to track the observed object. The presented approach connects frequency analysis with probabilistic estimation resulting in a feedback between both components, which approximates the real position in real-time.

2.2 Related Work

The problem to be solved can be described as a *motion capturing* problem with active LED markers [Barca et al., 2008]. Therefore many industry standard solutions can be listed, that implement such a capturing with active LED markers, but these solutions are designed to record data, which can be analyzed later. In contrast, my robot perception approach provides *instantaneous* position data usable for navigation.

As will be discussed later, there are certain advantages of high sampling rates in conjunction with active LED based tracking. In [Perry, 1990] for example, an extremely high sampling rate of 10,000 fps is used. While this is a system for use in high dynamic situations, industry standard systems work at frequencies around 50 Hz (Hertz) up to 1,000 Hz as a maximum. Furthermore [Schepers, 2009] states that there are professional industry solutions working with active LED markers. For example, *Optotrak*[3] is mentioned.

Another area containing similar problems are medical applications. To position a patient on full body scanning units like the Magnetic Resonance Tomograph (MRT) or the more widely known Computer Tomograph (CT), active LED markers are used because of their stability against environment lighting [Westermann and Hauser, 1996]. For reasons of higher accuracy, mainly infrared LEDs are used. This excludes a large amount of clutter in the captured images. A good overview about the state-of-the-art in on-line rigid body tracking is given in [Jansen et al., 2007].

Tracking in general can be done with many different approaches. As described by the excellent summary about this topic given in [Allen et al., 2001], what I am doing is optical tracking, in more detail ray measurement, using Charged Coupled Devices (CCDs) combined with active targets in an “outside-looking-in” (fixed camera with markers on the moving object) configuration. Following from that I am only looking at one small sub-area of tracking in general. Alternative approaches range from acoustic measurements over magnetic fields to inertia sensors and of course hybrid systems containing combined approaches.

A system which is more clearly related to my work is described in [Koch et al., 2008]. This paper describes a system for matching multiple camera views to a global coordinate system. Supporting my conclusions regarding the requirements for the proposed problem, Koch also tries to implement a fast, simple and lighting invariant marker pattern. He uses an LED at frequencies about 7 Hz and identifies the frequency by Fourier analysis. In Section 4.3.5 a comparison between Koch’s approach of segmentation, and the method used in this thesis is given.

In addition [Yamazoe et al., 2004] describes an even more similar system. In this approach flashing LEDs are again used to calibrate multiple camera perspectives. In contrast to Koch, Yamazoe et al. solve the correspondence problem directly as is done in this thesis and present a calibration method not only for static scenarios, but for complete video scenes. They encode not only identity information, but also time-stamps into the blinking pattern, allowing video streams to be exactly synchronized later-on. As the system should also be used in motion, stochastic means are used to predict linear motion. Analogous to my method of pose estimation, they also use homographies of planar feature arrangements to estimate relative poses.

[3]See Northern Digital Inc., `http://www.ndigital.com` for details

3 Basics

This section describes the theoretical aspects of the methods used throughout this thesis. For each of these methods a short introduction is provided to give the required background information. In Section 4, the algorithms are pursued in the context of the work presented in this thesis, based on the information presented in this chapter. Below a short list of used symbols is provided for orientation.

3.1 Frequency Analysis

For the described method of detecting LEDs by flashing patterns, a means of translating the sequences of pixel intensities into a binary blinking code is needed. To realize this, I initially began with an approach based on HMMs. This approach turned to be inappropriate for real-time purposes due to the computational complexity of the final algorithm, which is necessary to extract the desired information from the model's data. I therefore shifted from the HMM approach and selected a standard mechanism for frequency analysis using the Fourier transform. In the following section, I introduce both variants' theory.

3.1.1 Hidden Markov Models (HMMs)

According to [Rabiner, 1989], HMMs were developed in the late 1960s, but did not become popular until the middle of the 1980s. This statistical method is based on Markov Chains and allows to estimate the current state inside a model, for which certain state-transitions are defined. These states are hidden behind observations. Every time an observation is made, the "hidden" model performs a transition between two states. To allow a good estimation of the current hidden state, each problem has to be modeled using the different attributes a model provides. An HMM is formally defined by a 7-tuple $H = \{S, N, A, V, M, B, \pi\}$. The elements of this tuple are described in Equations (3.1) - (3.7).

$$\textbf{Hidden States} \quad S = \{s_1..s_N\},\ s_t = \text{state at time t} \tag{3.1}$$

$$\textbf{Number of States} \quad N = |S| \tag{3.2}$$

$$\textbf{Transition Probability} \quad A = \{a_{ij}\},\ a_{ij} = P[s_{t+1} = s_j \mid s_t = s_i] \qquad {\scriptstyle 1\leq i,j\leq N} \tag{3.3}$$

$$\textbf{Observation Symbols} \quad V = \{v_1..v_M\} \tag{3.4}$$

$$\textbf{Number of Symbols} \quad M = |V| \tag{3.5}$$

$$\textbf{Emission Probability} \quad B = \{b_j(k)\},\ b_j(k) = P[v_k \text{ at } t \mid s_t = s_j) \quad {\scriptstyle 1\leq j\leq N, 1\leq k\leq M} \tag{3.6}$$

$$\textbf{Initial Probability} \quad \pi = \{\pi_i\},\ \pi_i = P[s_1 = s_i] \qquad {\scriptstyle 1\leq i\leq N} \tag{3.7}$$

After these elements are defined, they describe how the respective process works. Basically, this is a graph of states S with transition-probabilities A between each pair of states, and observation symbols V as input with a predefined probability B of observing a given symbol while in a particular state. Through this a trellis is spanned, which later on can be traversed respecting the sequence of symbols observed.

All probabilities must be chosen manually in order to match the characteristics of the process to be modeled. This includes the probabilities π_i which describe how likely it is that a process based on this model starts in a state s_i. Having built a suitable primary model, algorithms like Baum-Welch [Baum et al., 1970] can be used to refine the probabilistic model.

After the selection of the set of states and symbols along with some reasonable probabilities, the process modeled by the HMM can start. This means, the process begins in some state s_i, which is unknown, with probability π_i. The only known element is the symbol v_i, which is observed on the next and all consequent observations. By this, it is possible to infer, based on the data of the model, which state the model was in when the process started and which path was chosen afterward. This traversal is calculated by the Viterbi Algorithm [Forney, 1973]. It tries to find the best state sequence $P = \{s_1, s_2, \dots, s_t\}$ for a given observation sequence $U = \{v_1, v_2, \dots, v_t\}$. For this purpose the algorithm tries to maximize a score $\delta_t(i)$ defined in Equation (3.9). This score denotes the highest probability along a path P having length t. After t observations from U it ends in state s_t. The tuple λ defined in Equation (3.8) resembles the set of all probabilities in a model H [Rabiner, 1989].

$$\textbf{Model Probabilities} \quad \lambda = \{A, B, \pi\} \tag{3.8}$$

$$\textbf{Viterbi Score} \quad \delta_t(i) = \underset{s_1, s_2 \dots s_{t-1}}{max} P[s_1, s_2, \dots\, s_t = i\,;\, v_1, v_2, \dots\, , v_t \mid \lambda] \tag{3.9}$$

3.1.2 Fourier Transformation

The following section provides an introduction to the usage of Fourier transformation, named after Jean Baptiste Joseph Fourier. It is a standard tool in Digital Signal Processing (DSP), which is elaborately explained in the DSP-Guide [Smith, 1997]. Much of the information presented here is based on that text, adapted to fit the needs of a rough overview for understanding the later application of the transform.

The main feature of this technique is the decomposition of a signal into sinusoids. Fourier proved that any periodic function of arbitrary complexity can be expressed as weighted sum of cosine and sine waves. In this thesis, discrete data in form of intensity values is used, thus the special form of the Discrete Fourier Transform (DFT) applies to the problem. For this type of Fourier transform it is assumed that there is a distinct number of discrete input values recorded over a certain period of time. By this a signal's wave form can be approximated depending on the sampling rate used. Such a signal can be seen in Figure 3.1. Furthermore a sampled signal in the context of a DFT is assumed to repeat towards negative and positive infinity.

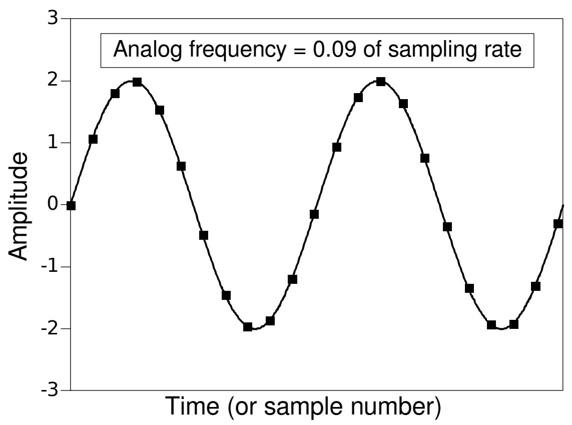


Figure 3.1: This graph shows a discrete periodic signal. Each square is a measurement or sample at the specific point in time which discretises the wave signal. [Smith, 1997, p. 190]

In general this number of n discrete input values $x_0 \ldots x_t$ is named the *time domain*. The Fourier transform now converts these n signals into the corresponding amplitudes of $\frac{n}{2}$ sine and $\frac{n}{2}$ cosine waves using Equations (3.10) and (3.11). Those are the basic equations for the DFT, which define the frequency-domain value in *rectangular form* in Equation (3.12), as sum over all considered frequency fractions. In an N-point DFT, the frequency k runs from $\left[0 \ldots \frac{N}{2}\right]$ while i takes values from $[0 \ldots N-1]$.

$$\textbf{cosine} \quad c_k[i] = cos\left(\frac{2\pi ki}{N}\right) \tag{3.10}$$

$$\textbf{sine} \quad s_k[i] = sin\left(\frac{2\pi ki}{N}\right) \tag{3.11}$$

$$\textbf{Rectangular} \quad X_k = \frac{1}{N}\sum_{n=0}^{N-1}\left[x_n\left(cos(\frac{2\pi kn}{N} - j\, sin(\frac{2\pi kn}{N})\right)\right] \tag{3.12}$$

This representation of the input is referred to as the *frequency domain*. It is important to mention that the frequency domain is only another representation of the same information. The process of transforming a signal from the time-domain to the frequency-domain is called *Fourier analysis*. Once the frequency domain signal is calculated, it can be converted back into the time-domain using the inverse Fourier transform (*Fourier synthesis*), without any loss of information. The loss of accuracy through floating point arithmetic is neglected in this statement.

After transforming the input signal into the frequency domain, a statement about the magnitude and phase of a certain frequency inside the signal can be made. However, as mentioned above the Fourier transform provides a set of amplitudes of both sine and cosine waves. This is called the rectangular notation. As the two trigonometric functions are only shifted in phase, both of them contain parts of the signal for a particular frequency. In order to combine them, it is necessary to use the polar representation of the frequency domain values.

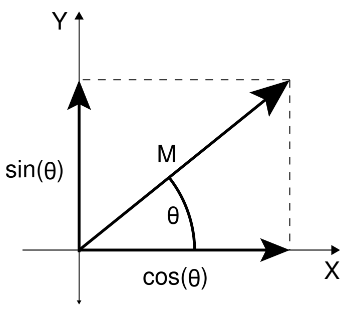


Figure 3.2: The picture shows the polar representation of the *sine* and *cosine* values by the angle θ, and the magnitude M. The magnitude corresponds to the signal strength, while the angle θ represents the signals shift to the corresponding wave.

As shown in Figure 3.2, a pair of sine and cosine amplitudes can be parameterized by a magnitude M and an angle θ, without any loss of information. This gives the possibility to measure the total amount of a certain frequency via the *magnitude* for a given input signal. The magnitude is of primary interest, as the *phase* only provides knowledge about where in the sequence of a particular frequency the capturing started, thus shifting the signal in time only affects θ. That is useful to determine the phase shift between the blinking of an LED and the cameras capturing rate. Equations (3.13) and (3.14) show the conversion from the rectangular to the polar representation.

$$\textbf{Magnitude} \quad M_i = \sqrt{cos_i^2 + sin_i^2} \tag{3.13}$$

$$\textbf{Phase} \quad \theta_i = arctan\left(\frac{cos_i}{sin_i}\right) \tag{3.14}$$

To explain the Fourier transform used in this thesis, the realm of complex numbers has to be entered. For mathematical correctness and easier representation, Fourier transforms are typically calculated via complex numbers [Smith, 1997, p. 577f]. In doing so, the gap between the sinusoids and complex values is filled via Euler's relation Equation (3.15). To avoid disambiguation for i as index, the engineering standard $j = \sqrt{-1}$ is used for complex numbers.

$$\textbf{Euler's relation} \qquad e^{x\,j} = cos(x) + j\ sin(x) \tag{3.15}$$

$$cos(x) = \frac{e^{x\,j} + e^{-x\,j}}{2} \tag{3.16}$$

$$sin(x) = \frac{e^{x\,j} - e^{-x\,j}}{2\,j} \tag{3.17}$$

As stated before, the Fourier analysis combines the sum of sine and cosine waves to form the frequency-domain value. Equation (3.12) shows the basic rectangular representation for a single information component in the frequency-domain. This component is the signal X_k constructed from the time-domain signal x_n. Using Euler's relation this can be turned to the polar representation for the complex DFT in Equation (3.18). Like n, the index k runs from $[0 \ldots N-1]$ and k is the index of a frequency-fraction $\frac{N}{2} \cdot f_s$ from the sampling-frequency f_s. In Figure 3.3, a frequency spectrum is shown. This visualizes the width of each "bin", which means one unit of *spectral density* $\frac{N}{2}$ as fraction of f_s. One such bin accumulates all frequencies' time-domain signal falling in its area.

$$\textbf{Polar } X_k = \frac{1}{N} \sum_{n=0}^{N-1} \left(x_n \, e^{\frac{2\pi k n j}{N}} \right) \tag{3.18}$$

Equation (3.18) builds the base of the Fourier analysis usually performed by scientific algorithms. It is, for instance, used in the library called Fastest Fourier Transform in the West [Frigo and Johnson, 2005] (FFTW). This library is internally utilized by the GNU-Octave software [Eaton, 2002], which I used to verify my results. Note that the equation used in the FFTW-Library is not normalized, meaning that the above added factor of $\frac{1}{N}$ is missing (see [Frigo, 2003] for details on the incorporated equations). In consequence, the factor is applied later during normalization for the segmentation described in Figure 4.3.4. Special about this implementation in FFTW is the precise way the equation is solved.

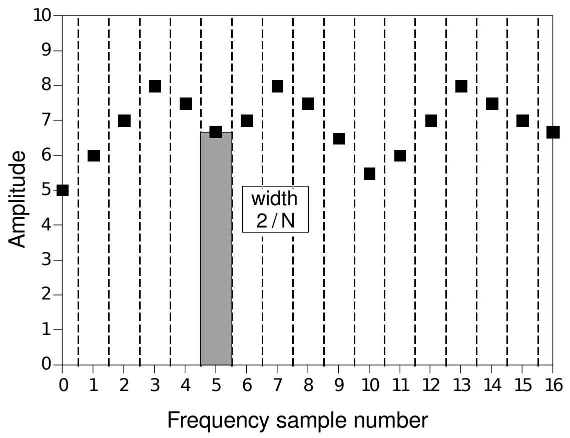


Figure 3.3: Example of a frequency spectrum. Each element in the spectrum covers $\frac{N}{2} \cdot f_s$ where f_s is the sampling rate and N is the resolution of the DFT [Smith, 1997, p.156]

As the name of the FFTW-library already quotes, the Fast Fourier Transform [Cooley and Tukey, 1965] (FFT) is calculated, which is a fundamental algorithm in DSP to accelerate the calculation of Fourier transforms. Although this is a really fast and robust implementation, there is the restriction of being forced to calculate the entire DFT-width, meaning all bins of the frequency domain. Of course this is no problem if all of them are needed, but in my case this is not required and slows down the algorithm. Depending on the way a frequency spectrum is interpreted, not all bins might be of interest.

3.1.3 Sliding Discrete Fourier Transform (S-DFT)

In order to split the Fourier Analysis binwise, there are several possibilities. For computational reasons, I decided to use the S-DFT introduced in [Jacobsen and Lyons, 2003], or rather the alternative formula introduced in [Jacobsen and Lyons, 2004]. Mentioned therein is also an alternative, computational superior, solution supported by the Goertzel Algorithm. However, the *Sliding Goertzel Algorithm* is implemented as *z-Transform* (another kind of signal transform in DSP), I chose the S-DFT with regard to the given time frame of the thesis, taking the small loss of one real multiplication per calculation.

The described algorithm relies on the *circular shift property* [Oppenheim and Schafer, 1989, 8.87, p. 567]. By this, we can express the spectral component value of a bin using its previous value, as well as the first and current input signal. This relation is expressed in Equation (3.19). It is the equation for the k^{th} spectral component bin. At this, $X_k(n)$ means the current new bin value, whereas $X_k(n-1)$ stands for the previous ones. Accordingly, the x values refer to the first registered input signal and the new observed one.

$$\textbf{Spectral Bin Value}\quad X_k(n) = X_k(n-1) \cdot e^{\frac{2k\,i}{N}} - x(n-N) + x(n) \qquad (3.19)$$

Finally, the computation for one new spectral component needs only one complex floating point multiplication and two real additions. This is one of the key features, which makes the S-DFT computation attractive compared to the HMM methods[4].

Despite the advantage of single bin computation, there is one drawback to this method. The sequential calculation of the values makes it impossible to omit any intermediate calculations. This means, that for every new input value x_n, the corresponding spectral component bin value $X_k(n)$ with index k must be computed, as it relies on its preceding value. Still, the S-DFT can be computationally superior to the traditional FFT implementation if for an N-Point transform a new spectral value is desired every M^{th} frame and $M < log_2(N)$. This is due to the runtime $O(N)$ of the S-DFT being smaller than the $O\left(N\ log_2(N)\right)$ of the FFT in this case. In my case every new value should be calculated, and therefore $M = 1$ and $N = 30$, making the S-DFT the method of choice.

3.2 Camera Calibration

When dealing with photogrammetric processes, one has to model the parameters necessary for a mapping of world coordinates in 3D space to the image plane of the camera. These parameters will adjust the equations used for the transformation. They approximate the properties of the optics and the relative attitude between camera and the observed object. The used calibration tools are described in [Bradski and Kaehler, 2008, pp. 370ff].

[4]See Section 5.11.2 for further details on how this property is used for runtime optimization

The mentioned parameters used for a description of the camera setup divide into two groups. In machine vision, determining these *extrinsic* and *intrinsic* parameters is a basic problem. It is described as the *perspective N-point Problem* where N is the number of points whose coordinates in both world and image plane need to be known to calculate a transformation between them. Knowing both parameter sets describes the camera's attitude and provides the mapping of Equation (3.20).

$$P_{world} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto P_{image} = \begin{pmatrix} u \\ v \end{pmatrix} \tag{3.20}$$

At first, the *extrinsic parameters* describe the relative attitude between the camera- and the world coordinate system via a rotation (Equation (3.21)) and translation (Equation (3.22)) of the world coordinate system's origin. They are applied in Equation (3.23). Equation (3.21) contains both the 3×3 and the 3×1 version of a rotation matrix. While the 3×3 maps a vector to the rotated vector, the 3×1 *Rodrigues Vector* contains the same information but performs a different rotation. A single custom axis is created by its coordinates, while the norm of the vector defines the rotation around this axis. After all, it is just a compact representation of a 3D rotation. However, this is the best physical description to model simultaneous motion on all axes.

$$\textbf{Rotation}\ R = \begin{pmatrix} r_{[0,0]} & r_{[0,1]} & r_{[0,2]} \\ r_{[1,0]} & r_{[1,1]} & r_{[1,2]} \\ r_{[2,0]} & r_{[2,1]} & r_{[2,2]} \end{pmatrix} \mapsto R_{\text{rodrigues}} = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} \tag{3.21}$$

$$\textbf{Translation}\ \vec{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{3.22}$$

$$\begin{pmatrix} x_m \\ y_m \\ z_m \end{pmatrix} = R \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \vec{t} \tag{3.23}$$

The Intel Computer Vision library (OpenCV), which was used for matrix operations, also models the lens distortion. In order to apply the particular distortion modifiers, the coordinate dimension is reduced from 3D to 2D coordinates in Equations (3.24) and (3.25). Subsequently the distortion can be included. OpenCV models radial translation via the parameters $k_{1\cdots3}$ depicted in Equations (3.26) and (3.27). Moreover it incorporates tangential distortion by parameters $p_{1\cdots2}$ in Equations (3.28) and (3.29).

$$u' = \frac{x_m}{z_m} \tag{3.24}$$

$$v' = \frac{y_m}{z_m} \tag{3.25}$$

$$\textbf{Radial x-Distortion} \quad u_r \approx u' \cdot \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{3.26}$$

$$\textbf{Radial y-Distortion} \quad v_r \approx v' \cdot \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{3.27}$$

$$\textbf{Tangential x-Distortion} \quad u_t \approx u_r + \left(2p_1 y + p_2 \left(r^2 + 2x^2\right)\right) \tag{3.28}$$

$$\textbf{Tangential y-Distortion} \quad v_t \approx u_r + \left(p_1 \left(r^2 + 2y^2\right) + 2p_2 x\right) \tag{3.29}$$

$$\text{where} \quad r^2 = u'^2 + v'^2 \tag{3.30}$$

Finally, the principal point c_x, c_y and the focal distance f_x, f_y for both image dimensions must be added. For this purpose Equations (3.31) and (3.32) do the initially discussed mapping of Equation (3.20)

$$P_{image}(x) \quad u = f_x \cdot u_t + c_x \tag{3.31}$$

$$P_{image}(y) \quad v = f_y \cdot v_t + c_y \tag{3.32}$$

3.3 Random Sample and Consensus (RANSAC)

The Random Sample and Consensus (RANSAC) algorithm [Fischler and Bolles, 1981] is a method to sort out outliers from a set of points belonging to a model. In general, every mapping from one set of values to another can be considered as a model. Therefore a simple example for a model is the equation for a line in space, which defines a mapping from a set of x values to a set of y values. In this model the slope m and the y-axis intercept n are the parameters of the model equation $y = mx + n$.

Given a set of noisy y values, the RANSAC algorithm is used to guess the model's parameters matching the provided results best. In order to do this it repeats several steps while approximating the optimal result in each step.

1. Select the minimal necessary number of data points to calculate the model. In case of a line in 2D space those are two points.
2. Calculate the model parameters.
3. Calculate the number of other data points whose distance to the model is smaller than a threshold t. The number of points propagated here is the *consensus set*.
4. If the size of the *consensus set* is above a threshold d, a fitting model was found.
5. Repeat steps 1 to 4 for k iterations and remember the model with the best fit for the observed data.

RANSAC represents a statistical approach to sort out outliers. Data points not fitting the calculated models will get sorted out quickly, so that only hypotheses explaining many points of the observed data will be kept. Of course the performance of the algorithm is dependent on the quality of the scoring system used to weight the model parameters against each other. In case of the above mentioned example, the scoring system is the Euclidean distance of a data point to the line. Additionally the parameters t, k and d have to be chosen reasonably to gain a quick, accurate model with the least effort. The accuracy is dependent on the value of k, as the quality of the estimation increases as the amount of iterations increases. However, each iteration also consumes arbitrary time, dependent on the used model equation. This means that the more iterations are used, the better the accuracy gets, but the more time is consumed by the process. The complexity of the model equation therefore regulates how good the algorithm performs provided a static time slot.

3.4 Particle Filter

Probabilistic means in computer science algorithms mostly rely on the *Bayes rule* in Equation (3.33). This rule grants the possibility to express the *posterior probability* $P(x|y)$ of an event y occurring in state x, using the likelihood $P(y|x)$, and the prior probability $p(x)$, which expresses the likelihood that a particular state will occur. In the following I want to give an introduction to the concept of a particle-filter using descriptions and notations from [Thrun et al., 2005].

$$P(x \mid y) = \frac{P(y \mid x)P(x)}{P(y)} \tag{3.33}$$

Understanding x as the current state of knowledge, and y as the new data observed, this means one can estimate the probability of a certain state, knowing only the probability of the occurrence of certain data. Algorithms working based on this assumption are commonly described as *Bayes Filters*. The principle behind these algorithms is to estimate the *belief* of a dynamic system at time t, $bel(x_t)$. In order to do so there are two steps performed by the filter. First, the *update rule* defines the transition from $bel(x_{t-1})$ to $bel(x_t)$. It does this by incorporating a control statement u_t, which causes a transition from state x_{t-1}, resulting in a new *predicted* state $\overline{bel}(x_t)$. This state expresses the belief, that the system really is in state x_t. Subsequently the second step named *measurement update* follows. In here, the belief $\overline{bel}(x_t)$ is assessed considering a particular observation z_t (*measurement update*). By recursive application of these steps the filter can give an estimate of the current belief, meaning the assumed state of the system.

Replacing the former symbol y of Equation (3.33) by the current observation z_t and conditioning the whole equation on the observations and measurements of all previous state transitions $(z_1 \dots z_{t-1}, u_1 \dots u_t)$ results in Equation (3.34).

$$P(x_t \mid z_1 \dots z_t, u_1 \dots u_t) = \frac{P(z_t \mid x_t, z_1 \dots z_{t-1}, u_1 \dots u_t)P(x_t \mid z_1 \dots z_{t-1}, u_1 \dots u_t)}{P(z_t \mid z_1 \dots z_{t-1}, u_1 \dots u_t)} \tag{3.34}$$

If now, it is assumed that every current state contains all knowledge induced by previous state transitions, we can express Equation (3.35), what is called a *Markov assumption.* In particular what is modeled here, is a *first order Markov process.* It derives from *Markov chains* named after Andrei Markov and contains the above assumption [Russell and Norvig, 2003, p. 539]. However this assumption is only a simplification, which may be violated by several special conditions, like for instance strong effects of the environment not modeled in u_t [Thrun et al., 2005, Ch. 2.4.4]. Together with an initial state x_0, which is believed to be true, the filter can guess all subsequent states.

$$P(z_t \mid z_1 \dots z_{t-1}, u_1 \dots u_t) = P(z_t \mid x_t) \tag{3.35}$$

Based on the initial ideas presented there are several options to estimate processes with probabilistic filters. The class of *Gaussian Filters* can estimate processes hidden behind Gaussian noise distributions and are generally the first choice for real-time applications as they feature fast calculations. One example is the *Kalman Filter* [Kalman, 1960]. Nevertheless I decided in favor of the *particle-filter* . The *Monte Carlo* methods used by this algorithm go back to [Metropolis and Ulam, 1949]. This is why the method is also called Sequential Monte Carlo (SMC). The approach has the advantage of being able to model arbitrary distributed environmental effects on the target measurement. Secondary in comparison to Gaussian filters their implementation is considerably straight forward and easy to extend. The latter is also the reason for a trend towards this method in probabilistic robotics. At last the approach benefits from its modularity, which means that the quality of the final estimation can be raised or lowered using more or less computation time. Analog to the presented RANSAC method this grants the possibility for a quality/performance trade-off.

The name of the particle-filter comes from the way it approximates the real state of a system. A particle describes one particular hypothesis of the current state. Given a robotic setup this is mostly one possible set of position data, which may be true or not (e.g. position $p(x, y)$ for a piece on a chessboard). Those particles are generated from the initial state of the filter by adding noise to those measurements and performing the implemented motion (cf. update rule). Afterward the generated particles are rated using measurements from the filters input data (cf. measurement rule).

At this, another important step of contemporary particle-filters is brought to bear, the *importance sampling* [Smith and Gelfand, 1992]. This process is described by Figure 3.4. In there we see the target probability distribution f in search, which shall be described by means of a second probability function g, which is observable. On the bottom it is shown how f can be approximated with samples of g, which were already weighted at this point. It is important that g will not produce zero values for values where f does not do so. In order to restart the steps of the particle-filter now the weighted particles are *resampled.* This means out of the weighted particles the same number of particles is drawn with respect to their individual probability. Hereafter all particles have the same weight and the cycle begins again. That causes the filter to throw away hypotheses, generated from the motion update, which do not fit to the observations made and will generate more particles at positions where both filter steps support each other. Therefore the filter will converge towards the real position with an accuracy defined by the number of particles used.

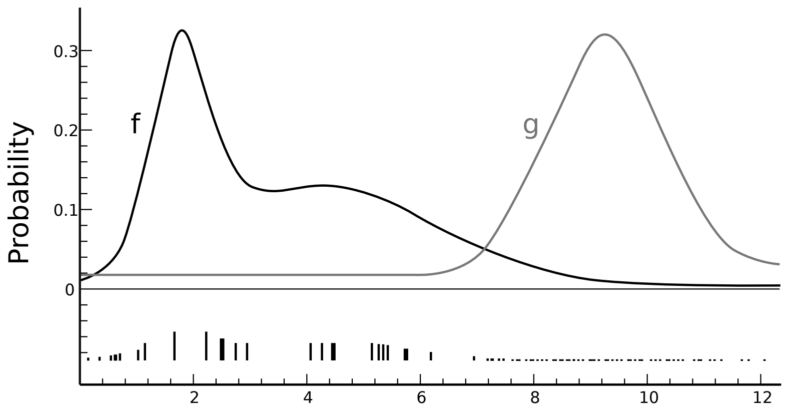


Figure 3.4: Probability distribution f is in search, while sampling is only possible from the second distribution g. The lines at the bottom show differently weighted samples (particles) approximating f (Image based on: [Thrun et al., 2005, p. 101]).

The exact method used for resampling in this thesis is a method known as *low variance sampling* [Thrun et al., 2005, p. 109ff], also called *Stochastic Universal Sampling (SUS)* [Baker, 1987]. The effect is shown in Figure 3.5. First, all weights of the particles are summed up, corresponding to the whole circle. Second, the average weight $\overline{w}$ of all samples is calculated, which is the distance of the arrows. Beginning at a random number $w_r = rand(0 : \overline{w})$, the sampling begins by choosing the particle at that point in the sequence. Afterward $\overline{w}$ is added to the current weight w_c, and the next particle p_n matching the condition of Equation (3.36) is selected.

$$w_r + \sum_{i=0}^{n}(w_i) > w_c \tag{3.36}$$

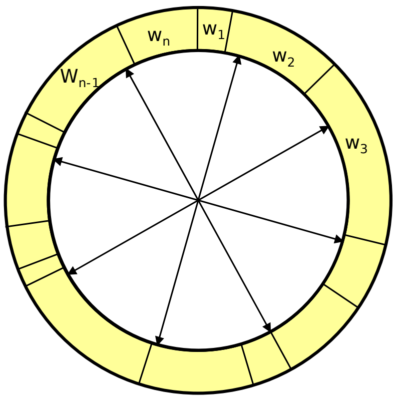


Figure 3.5: The pie symbolizes the sum of all particle's probabilities. Each individual particle is represented by a segment of size w_i according to his fraction of the weight-sum. The arrows depict the positions where a particle is selected. An arrow pointing to the segment of a particle means, that he will be contained in the new sample. (Image Source: [Burgard et al., 2009, p. 23]).

The technique has several advantages over choosing random particles. If for example all samples have the same weight, this method simply maps all particles to the next sample. Furthermore the algorithm can be implemented with runtime $O(N)$.

4 Approach

In this section I want to introduce the methods I use in order to solve the problems described in Section 1.2. It is about how to apply the theory introduced in chapter 3 to fill the gap between a blinking LED on the input side, and a final pose as output. In the beginning a short overview will outline the whole framework. Thereafter the approach to the system is incrementally developed, beginning with the *segmentation* in Section 4.3, which defines the interpretation of pixel intensities in the image. Among the actually used DFT, the ideas behind the previously examined segmentation with HMMs are explained. After that, some specific problems of the DFT method used are discussed in Section 4.4, which were encountered on the way to generate an initial pose with the approach. Finally, Section 6.3 describes how the initial pose is refined using probabilistic methods. This approach makes pose tracking in motion possible. At last, I shortly introduce the hardware used and the consequences for the final system. To separate names of code objects from floating text, a `mono spaced font` is used. Furthermore the names of the thesis' software modules, described as subsections in Section 5, are highlighted (e.g. `DFTSampler`).

4.1 Overview of the Approach

The planned system assumes a split setup with two components. On the one hand, there is the fixed camera driven by the developed software as active, pose searching part. On the other hand, there is a pattern of planar LEDs, blinking at a given frequency attached to the observed object. Connected via FireWire (IEEE 1394 interface), the camera is recording frames at 30 Hz.

In a first step, the raw 8 bit monochrome (shades of gray) frame is analyzed pixelwise over time, to search for a certain frequency. According to a final threshold, clusters of pixels possibly belonging to a certain LED are built. In the subsequent step, the system chooses one combination of pixels suitable for the observed clusters. This is done by connecting the known geometry of the LED pattern with the observed pixel positions. Finally, they are combined to a perspective model of the attitude between camera and object in the scene. This works for static scenes.

However, as soon as the object moves, it is impossible to gain frequency information, without knowing the parameters of the movement. Instead, this method initializes a modified pose estimation, which provides the necessary motion information. The estimation algorithm starts to guess the movement of the object, and interpolates the new LED-positions by the assumed position. This way, many hypotheses for the motion of the observed object, in this case the cart with mounted LEDs, get created. Afterward the interpolated LED-positions are used to provide a new LED image-position for the initial method of frequency analysis. At this point, a redundant cycle of supporting estimation is started, which is finally able to estimate the pose by using frequency information, although the pixels in new frames may not match those observed in the previous frame at the same coordinates.

Altogether, the planned positioning system should perform well enough to determine the position of an object in space, navigate in its direction and place a robot relative to the detected heading. As practical example for the accuracy, a robot placing himself relative to a cart can be considered. After moving behind the cart, the robot should be close enough to the right position to climb up.

4.2 System Overview

The system overview in Figure 4.1 explains the connections between the modules of the whole system. Herefrom arises the division of the system into three global steps, the *Acquisition*, *Segmentation*, and *Pose Estimation*. While the Acquisition is only an auxiliary step to get frames from either a camera device or an image from the hard disk, the pixel related *Segmentation* relates pixels to an LED based on whatever method is used, whereas the *Pose Estimation* is used to filter positional information in space out of the LED poses found. The particular modules used to perform those tasks are described further in Section 5.

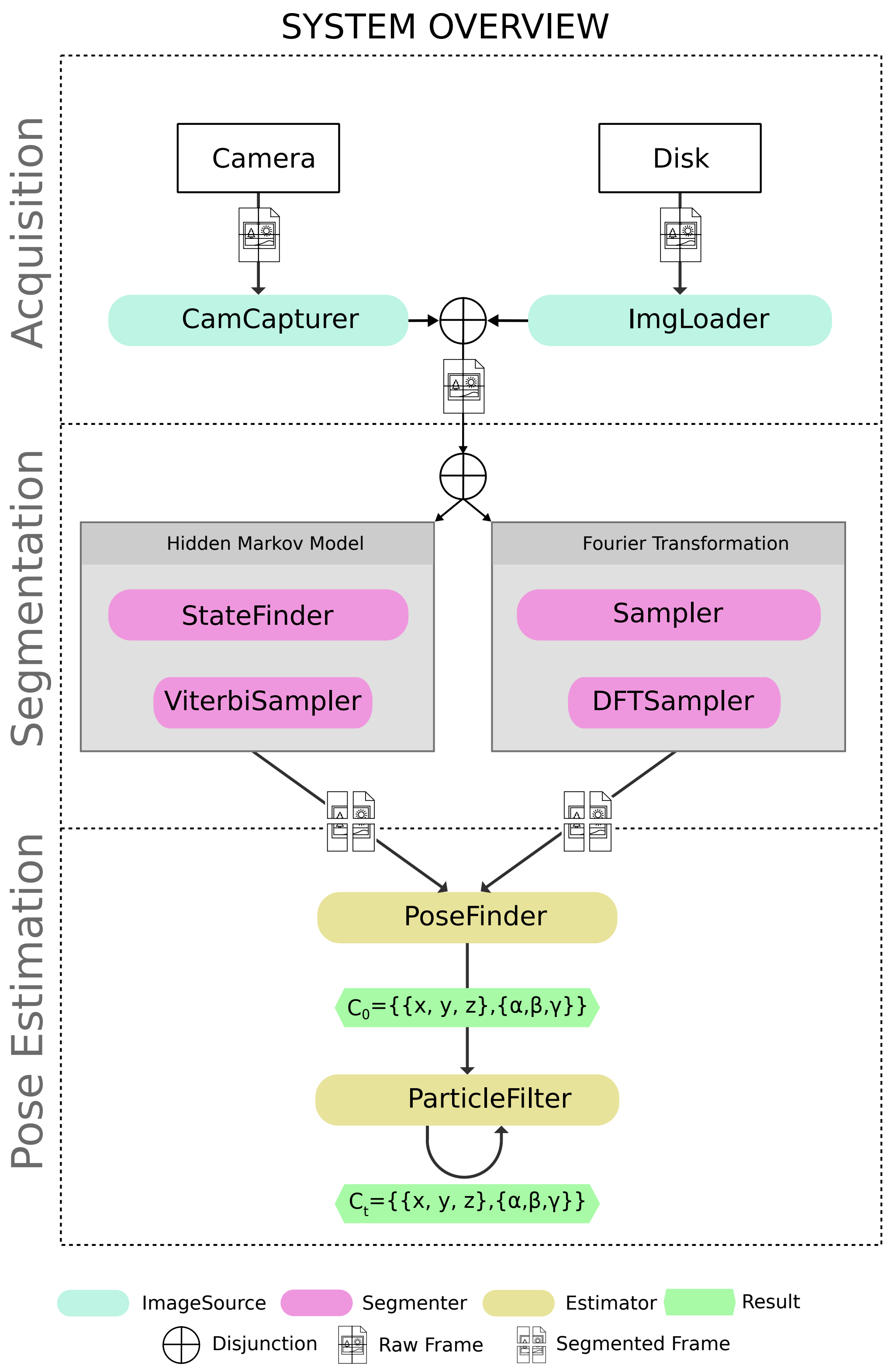


Figure 4.1: System overview showing the core components of the overall system

4.3 Segmentation

4.3.1 BLOB detection

To avoid the problems with defining color-value ranges described in Section 1.1, a completely different kind of image perception is used. For this purpose, the pixelwise intensity information received by the camera is not interpreted as directly usable value, but as part of a binary signal. This signal means, a lit LED represents the 1 state, while an unpowered LED corresponds to the 0 state. By doing so, the problem of categorization for a certain pixel changes from the decision to map a certain color to a color space (e.g. 255^3 for RGB), to a mapping on two states. Although this introduces an advantage, it raises several problems. In Figure 4.2 the subsequent image frames $F_0 \dots F_N$ on the time axis t show how the intensity value ι is interpreted as a sequence of values from a pixel on the same position in each frame. The result is an intensity pattern.

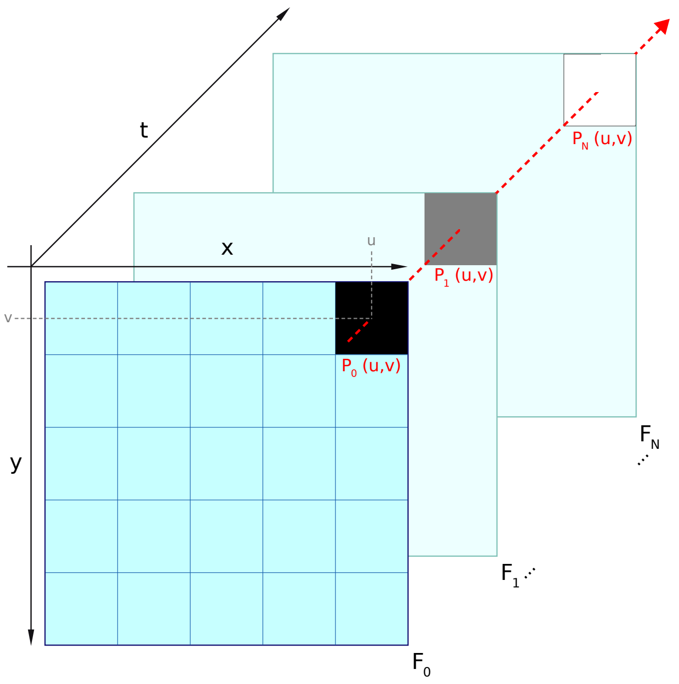


Figure 4.2: Illustration explaining how the information on the image plane are interpreted as continuous signal on the time axis t

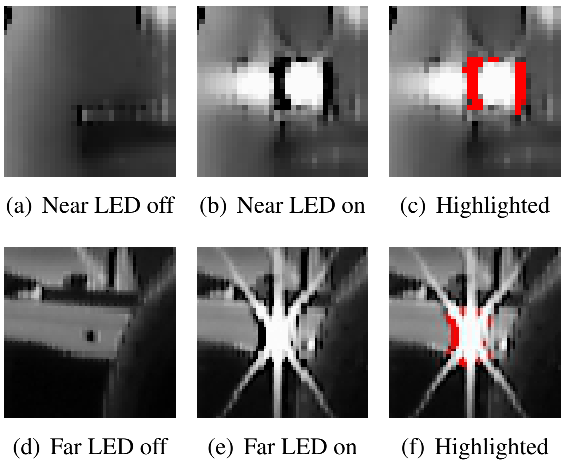
At first, the mapping of a pixel to either the *on* or the *off* state still requires a certain thresholding likewise done in more complex color spaces by look-up tables. A simple thresholding is not sufficient as the whole image context is reduced to a smaller range when considering 8 bit grayscale images. Because of this, many areas in the image may contain the same information as the lit LEDs, preventing the LEDs from being exclusively filtered. This can be overcome by treating sequences of images as the information to be processed, instead of only single frames. In other words, an LED is not by definition a light spot on an image, but a certain combination of light and dark spots that represent a certain pattern. This interpretation leads to a significant problem area: By interpreting *sequences* of images, we lose the possibility of classifying an object within a single frame. Therefore, the benefits of a more simple classification come at the price of sequential dependency. Nonetheless, this is not only disadvantageous, because when information is spread across many images, the quality of the information hypothesis of an object at a certain position is automatically averaged while in the case of color frame segmentation, this must be done manually with filters. In general, the drawback is outweighed by the fact that sequential grayscale data is not susceptible to many lighting problems, which aligns well with the goal of this work.

4.3.2 Blooming and Anti-Blooming

During the initial test records for the construction of HMMs two optical effects causing problems in segmentation appeared. When the blinking LEDs are recorded at a distance of only one or two meters, bright stars appear on the images. In general, nearby light sources appear bigger than their light emitting surface. This effect is dependent on the distance and the brightness of the light. It is called *blooming* and appears if the light sensitive units on the camera's CCDs exceed their maximum charge. In this case the superfluous charge is passed to the neighbored units, so they register light although there is no light source at that position. Additionally there is an optical phenomenon causing the star-shape, or *spikes*, how they are called in astronomy. The effect appears if wave forms hit surfaces and is called *diffraction*. The light slipping through the small apperture hole is thereby spread in the spikes' direction. Furthermore diffraction is caused by the grid arrangement of the light sensitive elements on the CCD. They work like a *diffraction grating* used in physics [Van Overschelde and Wautelet, 2005]. Due to the high power LEDs being used near to the camera, the mentioned *spikes* appear in the images (d) - (f) of Figure 4.5 at 1.5 m distance, whereas the effect is not visible in images (a) - (c), recorded at 9 m distance.

In addition, looking closely at the recorded light reveals a black shadow around the center of each LED. These shadows are visible in the images of Figure 4.3 and are caused by Anti Blooming Gate (ABG) firstplurals [Keenan and Hosack, 1989]. Figures (a) and (d) show the background of the LEDs, to verify that those pixels were lighter than black before. After the LED was lit (Figures (b) and (e)), there are black regions around the center of the light-cone. Those black regions are highlighted in (c) and (f) to show their positions.

The ABG-mechanism detects the blooming effect electronically by shifting charge between the sensor units in order to cancel the superfluous parts, meaning the charge above the sensitivity level of it. Because of this, there are some black values at the points where the charge was removed. The described effect entailed difficulties in detecting the light source of an LED. Problems occur when a rising intensity edge is expected by the LED being switched on, but the pixels affected by the ABGs perform the opposite change in intensity. The problem was finally solved by switching to the Fourier analysis as blinking pattern recognition. In contrast to the Hidden Markov approach this method does not expect a certain intensity to appear, but only a pulsating change. As in this case a rising and falling intensity edge exposes the same rate of changes per time, the different edges will reflect the same information though at different


(a) Near LED off (b) Near LED on (c) Highlighted

(d) Far LED off (e) Far LED on (f) Highlighted

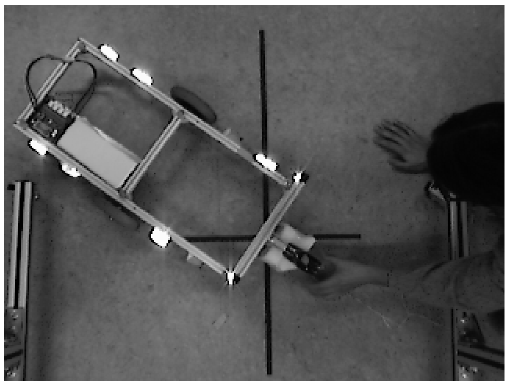
Figure 4.3: The effects of ABGs are shown by highlighting all pixels, which got black without reason. (a) and (d) show the background, while (c) and (f) show the same images as (b) and (e), but mark black pixels on the border of the light cone in red.

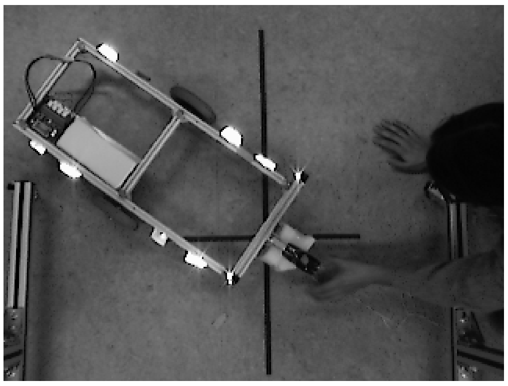
intensities. More recent CCD technologies have appeared which overcome this problem, but because the camera used in this diploma thesis lacks the newer technology this problem must be considered during the system design.

4.3.3 Application of the HMM

In a first step, I evaluated LED blinking using HMMs. The idea behind this is to understand the blinking pattern, its sequence, as a hidden state. Determining the exact position in this sequence, and also to distinguish different sequences containing the same elements, is not possible directly. Instead the intensity information ι from an image can be observed. These serve as indication for a specific state. Filtering knowledge out of dependent events is a classical task for HMMs. With the LEDs being switched on and off the blinking can be understood as binary pattern.

After investigating different types of HMMs, I chose to use a simple left-right model [Rabiner, 1989, p. 266], meaning each state has a transition to one other state but can also, with a very small probability, transition to itself. This "self transition" is a standard approach that compensates for inaccuracy between the model and the real process. To be independent from the intensity bias, I chose to use the intensity difference $\Delta\,\iota$ between two subsequent 8 bit frames $\Delta\,\iota$ as input symbols. The resulting difference frame F_{diff} from the subtraction $F_{\text{diff}} = F_t - F_{t-1}$ is an array of 16 bit values in the interval $[-255 : 255]$. Consequently the observation symbols V are based on a signed difference image between two subsequent 8 bit grayscale images. To visualize such an image I calculate a three channel image out of the 16 bit single channel values gained from the difference. The pixels with positive sign are drawn on the green channel, those with negative sign will receive an according blue value. An example is shown in Figure 4.4.


(a) Minuend


(b) Subtrahend

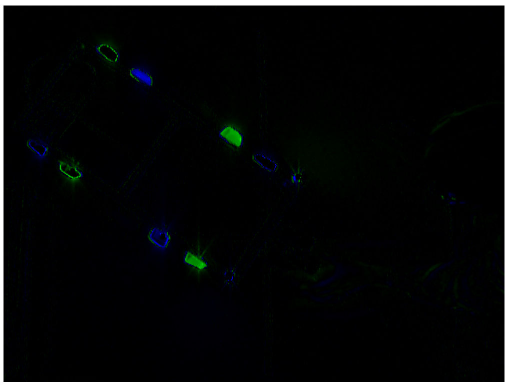
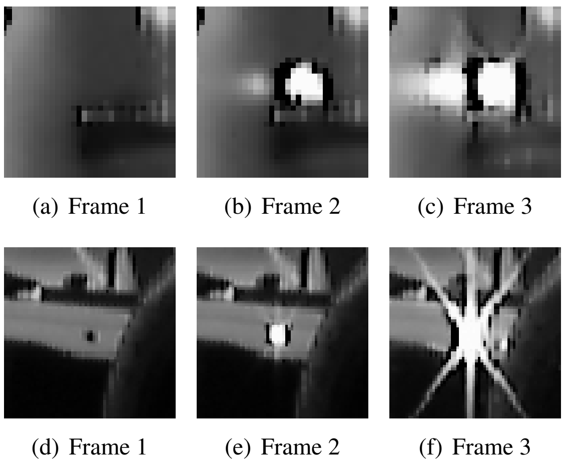

(c) Difference

Figure 4.4: The difference image in 4(c) shows the result of the operation (a)−(b). Blue marks pixels, which became darker while green pixels became brighter

The next step to explain how HMMs can be applied to my problem is to define the interpretation of pixel values. Using this as input signal, an HMM can be built to estimate the affiliation of a pixel to an LED or to the background. This is the most tangible information provided by HMMs, which I call the meta-state. Each LED is represented as such a state and is defined as an independent string in the model. In general, what is to be determined by the HMM is to which meta-state the observed ι sequence belongs. Moreover it is desirable to know at which position inside the blinking pattern the current observations are located. Figuratively spoken we want to know if the LED is being switched *on*, *off* or *zero-off* at the moment. The latter means that a switched off LED stays off. These three states are called intermediate-states and are the building blocks of each meta-state. Finally, there are the real hidden-states of the HMM as described in Section 3.1.1. These represent a certain $\Delta\,\iota$ value of a pixel at a distinct position in the blinking pattern. The term *hidden* thereby depicts that they cannot be observed directly. The position of a particular intensity value ι in the context of the HMM is not observable, instead we can only observe ι values and reason from their sequence to their position in the pattern. It becomes clearer how to imagine the connection between all different described states by looking at Figure 4.7. This figure also contains also an advanced concept described in the following.

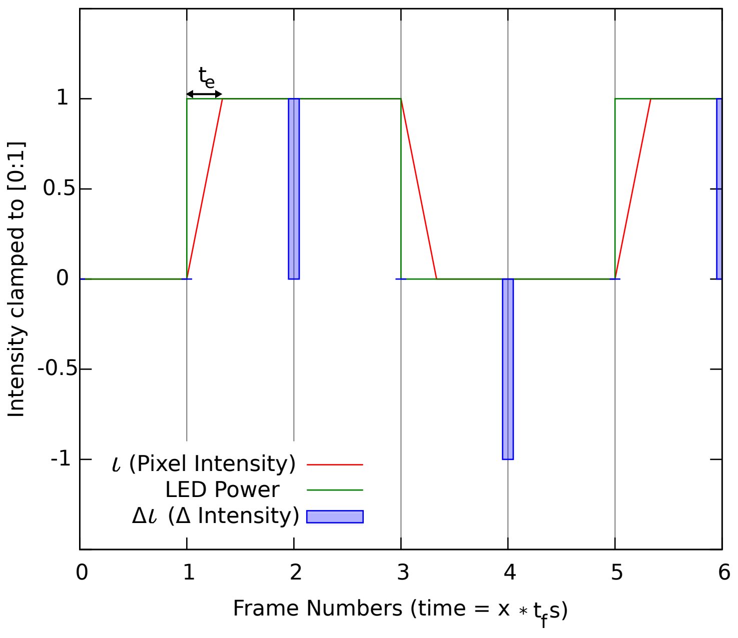

(a) Frame 1 (b) Frame 2 (c) Frame 3

(d) Frame 1 (e) Frame 2 (f) Frame 3

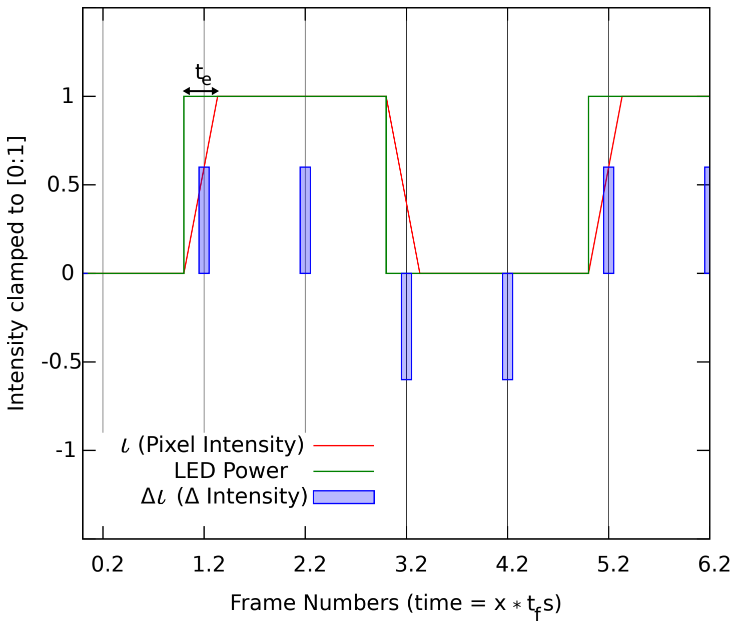
Figure 4.5: These are two 64×64 pixel parts of three subsequent frames in each case. They show an LED being switched on at ca. 9 m in (a)-(c) , and at 1.5 m distance in (d)-(d). After two frames it reached its full intensity. Any following frames in the *on* state looks the same as the third one.

This concept dissects the intermediate-state more finely. The reason is that it helps to overcome some practical problems caused by the camera synchronization issues, which were encountered during the implementation of the approach. Those problems are present due to the camera and LEDs not belonging to the same system and thus being asynchronous. Hence, this asynchronism has to be modeled by the system. To do this, it is necessary to be aware of the consequences for an intensity value in the resulting image. In Figure 4.5, there are three consecutive frames taken from an LED being switched *on*. One can see that in the middle of the LED there is light after only one frame, but on the border of the high intensity region, there are pixels which do not get bright until the second frame. This is caused by asynchronism. In Figure 4.6, the situation for one pixel in the center is illustrated. It is assumed that the LED is switched *on* instantly, stays two frames in the *on* state, and is switched *off* after exactly 2 frames. Furthermore we assume the camera, which has a frame rate $f_c = 30\,\text{fps}$ with an exposure time of $t_e = \frac{1}{100}\text{s}$ (second), is adjusted perfectly to provide the maximum grayscale value of 1 after t_e seconds. Each frame takes $t_f = \frac{1}{30}\,\text{s}$ time until it arrives at the system. Starting in frame 1, the LED is switched *on*. This is visualized by the green line. In contrast, the red line, which stands for the pixel's intensity ι, now rises linearly with the exposure time until it reaches full intensity after t_e. Vertical lines represent the point in time at which a frame is recorded by the system. In Figure 4.6(a), the frames are recorded exactly at the switching point of the LED. As a result, the pixel will have an intensity of 1 at frame 2. However, if the phase of the capturing points is shifted by $0.2 \cdot t_f = 6.\overline{6}\,\text{ms}$, the intensity is recorded during the rising edge of the intensity values. The difference value between two consecutive frames is drawn as blue bar centered on the frames. Resulting from the phase between the camera capturing and the LED phase, those values look completely different, but still model the same pixel.

- **Observation Symbols:** $\Delta\,\iota\;[-255 : 255]$
- **Hidden-States:** $\Delta\,\iota$ value at a distinct position in the blinking pattern
- **Intermediate-States:** An LED being switched *on*, *off*, or no light at all. Each intermediate-state has the horizontal length of two *hidden-states*.
- **Meta-States:** A combination of intermediate-states modeling a sequence of rising, falling or constantly zero $\Delta\,\iota$ signals. This sequence is unique for each LED so it can be identified by a hidden-state belonging to an intermediate-state, which belongs to the meta-state of this LED. Although the sequences consist of the same components of intermediate-states there will be separate copies for each meta-state. This is necessary as each intermediate-state may have different successor or predecessor states depending on the meta-state it belongs to.



(a) capturing phase 0



(b) capturing phase 0.2

Figure 4.6: Synchronization between camera and LED visualized for one pixel in the center of an LED-perception. Exposure time t_e is $\frac{1}{100}$ s while the camera captures at 30 fps. $\Delta\,\iota$ is considered between the current and the last frame.

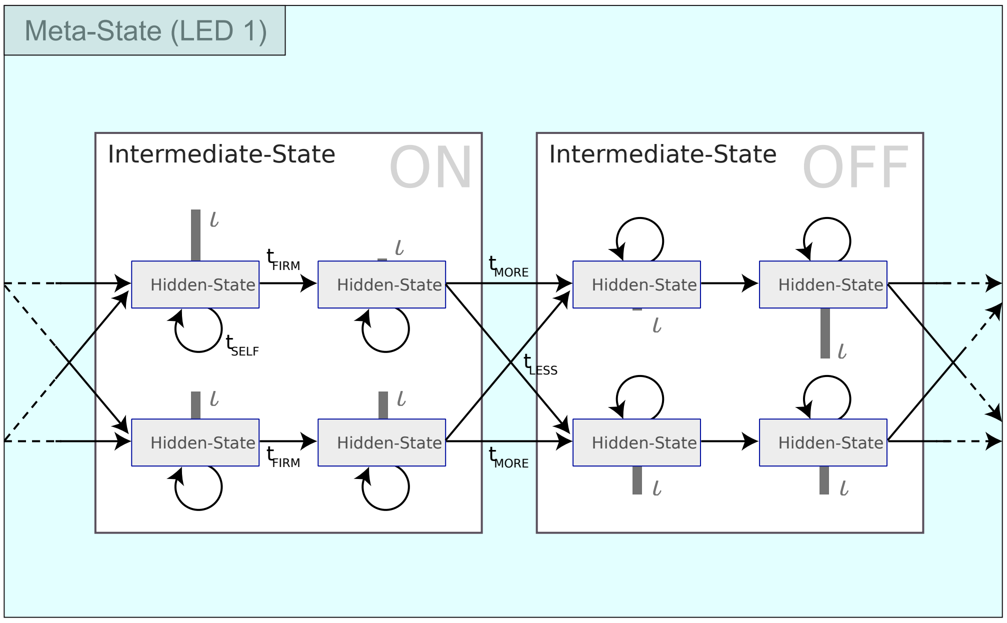


Figure 4.7: Example for a used HMM representing an LED being switched *on* and *off* repeatedly

Figure 4.7 shows the smallest example for an LED-HMM. The meta-state models an LED being switched *on* and *off*. Note the dashed arrows in front of, and after the state. They are connected with each other, but were chopped for better visualization. Inside this meta-state the two intermediate-states representing the LED switched *off* and *on* are visible. Finally, the intensity values ι are shown as gray bars. They correspond to the blue bars described in Figure 4.6.

To model the probabilities A for state transitions four levels of transition were introduced. SELF transitions have a very small probability of 0.02, in opposition to FIRM, which describes an almost certain probability of 0.98. Shifting phases is possible but seldom, thus it gets a small probability LESS of 0.05, while the normal transition stays in the current phase with probability MORE valued 0.95. For the emission probabilities B, three kinds of functions were used. If a specific value is expected a Gaussian is centered on this value, otherwise a rising or falling logistic function describes the tendency of the expected ι.

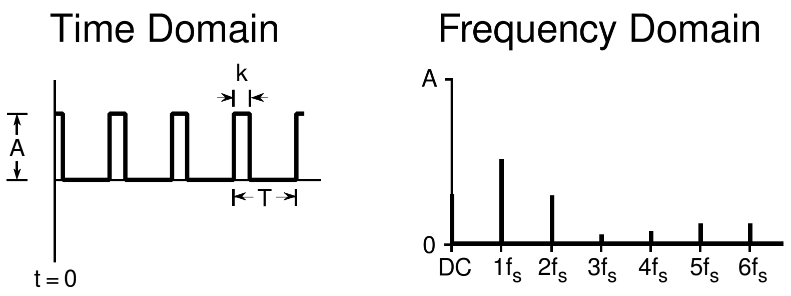
The method to extract the information about the current state of a HMM is the Viterbi Algorithm mentioned in Equation 3.1.1. This algorithm traverses the HMM given a $\Delta\,\iota$ value sequence $Q = v_0 \dots v_n$ and reports the most probable path of states exhibiting Q. In fact it calculates every path's probability, so the most likely path can be extracted out of this result. Furthermore, it reports a total probability for the traversal. According to [Russell and Norvig, 2003] the Viterbi algorithm has a linear runtime $O(n)$ where n is the input-sequence length. Although this is a linear run-time, it is important to look at the omitted constant factors. In each step, the algorithm checks the probability of each state S_i (pre) changing to S_{i+1} (post) observing the symbol v_i. Consequently, the runtime is dependent on the constant number of states $|S|$ resulting in $O(n \cdot |S|^2)$. As this additional factor of $|S|^2$ is applied pixelwise, it grows very fast. An initial implementation of an HMM for 6 LEDs, constructed from the intermediate-states discussed earlier, resulted in about 80 states. To avoid ambiguity, the signal patterns of the LEDs have to be disjoint in phase. This means the state combination on, off is the same as off, on, since it is not known when the algorithm starts to observe the pattern. Hence, the allowed combinations of intermediate-states result in a fast growth in length of the meta-states each, adding 4 hidden-states to the HMM as a whole. Summarizing, the introduced method proved to be not appropriate for pixelwise use. Otherwise the algorithm is very easy to parallelize, what made it popular in DSP. Research for improved implementations of the Viterbi algorithm revealed the "lazy Viterbi algorithm" [Feldman et al., 2002]. An optimal runtime of $O(1)$ is mentioned, while being not worse than standard Viterbi in the worst-case. However, the algorithm is fit for convolutional codes with only two states 0 and 1. Resulting from this, I searched for another method of recognizing blinking patterns, which is described in the following chapter.

4.3.4 Using the Fourier Transform

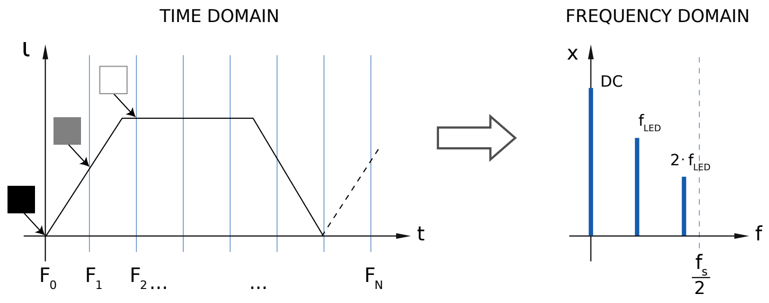
As reference implementation for the detection of LED blinking patterns I use Fourier analysis. Again, I want to begin with explaining the manner information is interpreted by this method. Starting on the raw 8 bit values of the input image, there is for each pixel discrete information about its intensity. Instead of using now $\Delta\,\iota$ to look for the change, each pixel's intensity value ι is interpreted as direct part of the LED's blinking signal. Unlike in the context of the HMM, where LED switch sequences are understood as binary patterns, they are now considered as frequency signal. More precisely they are samples of such a frequency at special points in time, those points where the camera took a picture. As such, they are discrete samples of their frequency. That can be determined by using Fourier analysis.

Interpreting the input image pixelwise means that for every pixel a frequency spectrum is kept. In order to calculate this spectrum, the sequence of 8 bit values for a pixel with coordinates $[u, v]$ (see Figure 4.2) over several frames is pushed on a circular buffer representing the time-domain signal. The frequency spectrum which is generated using the DFT indicates which frequencies are contained in the last N values for this pixel. Thus each pixel's intensity history is now converted to the frequency domain. At this point it is once again necessary to look at what can be inferred by this information.

In 8(a), the spectrum for a rectangular pulse function with parameters is visible. This is the function an ideal LED would provide. It is a *square wave* function with a certain y-offset. For an LED blinking at $f_{LED} = 5\,\text{Hz}$ at a sampling rate of $f_s = 30\,\text{Hz}$ its properties are: $A = 255,\ k = 4,\ T = 6$. The figure shows that the input signal on the left results in spikes on the right at each multiple of f_s. These spikes in the frequency-domain are the harmonics of the sampling rate. At this point I want to mention that the Fourier transform of a rectangular function is the *sinc function*. Resulting from this spectrum the first harmonic of the pulse function can be identified as frequency with the largest fraction of the frequency-domain signal. Though this is true for the signal shown in 8(a) where $k < k - T$, in the case of the 5 Hz LED then $k > k - T$. As a result the *on* portion of the signal is larger than that for *off*. Hence the 0- bin rises above the first harmonic. This is due to the fact that the zero- bin in the DFT represents the Direct Current (DC) part, the average value, of the LED signal.



(a) A pulse function in time-domain and frequency-domain



(b) Time domain in the context of pixel intensity

Figure 4.8: The figures show a pulse function in time and frequency domain. Figure (b) connects the intensity value ι with the time domain in the context of a Fourier transform. The ι values are adopted from Figure 4.2 to show the connection, the function is leaned on Figure 4.6 (Image source for (a): [Smith, 1997, p. 257]).

Therefore we omit the 0- bin and look for the first harmonic, which is the frequency that looks the most like the input signal and is also the frequency that should be found while observing an LED. To separate this frequency signal from the others, it is possible to just select the bin with the highest frequency fraction. Since the S-DFT is used, the calculation can be skipped. At this, we make the assumption that all LED's frequencies were selected so that the first harmonic of each LED is detectable by one of the DFT's bins. To reach this, for a sampling rate f_s of 30 fps, we have to use a $N = 30$ element wide DFT. This way, each bin corresponds to a fraction of the sampling rate, which is $\frac{f_s}{30} = 1\,\text{Hz}$. This means that each bin has a *spectral density* of 1 Hz, so every frequency registered in this bandwidth will cause this bin to rise.

Concluding it is possible to define a pixel as belonging to the frequency with the highest spectral bin except 0. However, this would lead to a lot of over-segmentation. It is necessary to apply a threshold for the signal strength, such that the center of an LED receives a good value, and reflections or blooming in the image are devalued. Furthermore the measure shall be invariant to changes in intensity. Hence it is not important which intensity value the blinking LED exhibits. To reach this, I use a simple equation, which embodies the relation between a pixel's frequency-domain value and the standard deviation of the intensity ι, shown in Equation (4.4). The resulting value obtained by applying this equation to a frequency-domain signal is called $\hat{X}$ in the following.

In order to force this normalized value to the $[0 \cdots 1]$ range, it is necessary to normalize the frequency-domain signal first with a factor of $\frac{2}{N}$. Hereafter, the signal has the same scale as in the ingoing time-domain. Using the absolute value $|X_{\text{norm}}|$ at this point causes the imaginary and the real part of the frequency-domain signal to collapse into a single real value. Finally the division by $\sigma(x)$ from Equation (4.1) causes the resulting value to be dependent on the standard deviation from the pixel's ι values. It utilizes the computational formula for the variance to minimize the effort to calculate the value in each new step of input.

$$\sigma(x) = \sqrt{\frac{1}{N}\left(\sum_{i=1}^{N} x_i^2\right) - \left(\frac{1}{N}\sum_{i=1}^{N} x_i\right)^2} \tag{4.1}$$

$$X_k = \sum_{n=0}^{N-1}\left(x_n\, e^{\frac{2\pi k n j}{N}}\right) \tag{4.2}$$

$$|X_k| = \sqrt{Real(X_k)^2 + Imag(X_k)^2} \tag{4.3}$$

$$\hat{X}_k = \frac{2\,|X_k|}{N\,\sigma(x)} \tag{4.4}$$

Using this normalization, the basic brightness of a pixel does not influence the normalization value as the standard deviation represents the power of the signal's fluctuation. By this, pictures of different brightness can be thresholded in the same way. Additionally the factor $\frac{1}{N}$ is present in each part of the equation, making it invariant in terms of the DFT-resolution N. Moreover Equation (4.5) represents the average ι for one pixel. In DSP his is also referred to as the DC component of a signal. As this term is included in $\sigma(x)$ in Equation (4.1), the measurement is also not dependent on the amplitude of a signal.

$$\overline{x} = \text{DC} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.5}$$

Nevertheless, there is one problem left. First, the blooming causing the characteristic "stars" around a nearby LED are not completely cut off. The reason therefore is the missing synchronization with the camera's shutter (see 4.3.3). As the center of the LED is exposed first, but can also be bright on a frame where the LED is not lit anymore, it seems to match the searched frequency less good than a pixel on the arms of the "blooming stars". The latter is an effect of partial exposure over a fraction of the current exposure time. Resulting from this, it is not possible to depict a perfect threshold to separate the LED core from those caused by blooming or reflections. In order to do so, the synchronization between camera and LED flashing has to be known, or the selection of a specific bin should not only depend on the highest bin of the first harmonic. Instead of this, the whole spectrum can be interpreted like a fingerprint of the LED searched. The following two chapters look in more detail at the problems arising from this approach and methods to deal with them.

4.3.5 Comparison with Koch's Segmentation

In an approach similar to my approach, Koch tries to detect blinking LED frequencies on a per-pixel basis to gain perspective knowledge from a picture. [Koch et al., 2008] In contrast to my approach Koch uses multiple cameras and is only interested in finding the same spot of an LED in each cameras image. Hence, his method uses multiple perspectives and geometrical constraints of more than one camera observing the same object. Furthermore Koch uses LEDs in the infrared spectrum. In general, he does not search for a certain arrangement of LEDs, but for one single light spot. Apart from that, Koch's work shares with this thesis the same method for segmenting the pixels. He analyzes sequential images and tries to find frequency information for a pixel by analyzing its intensity values $i_1 \cdots i_N$ with N being the resolution of his Fourier analysis.

The biggest difference between Koch's and my approach is the way that LEDs are chosen as belonging to a certain frequency or not. While I simply assume the largest bin value as the LED in search, he infers the affiliation to an LED as a metric over all bins of the frequency-domain. By comparing the observed combination of bin values as a *feature vector*, he calculates a similarity between the observed and the expected spectrum. In the following he evaluates the advantages of using the Euclidean or the Mahalanobis distance as measure between the stated vectors. His results show that the best rate of information while choosing a minimal DFT resolution $N = 8$ is reached at a blinking frequency of $f_{LED} = 7.5\,Hz$.

Koch's results show that it is not necessary to use a high number of bins, but also use a very limited number of bins with another metric of comparing the observed spectrum to the expected. However, Koch only uses that metric as a binary decision method to determine if an LED was observed or not. In his paper he does not examine how well his method works to separate different LED frequencies from each other. Nevertheless, his approach can be combined with my selective calculation of bins. By thresholding on the mentioned feature vector, and not only the highest or most likely bin value, aliasing effects can be incorporated in the separation process. This might help to separate pixels in the LED's core from those getting lit by the blooming effect (see Section 4.3.2).

Finally one distinct difference between Koch's implementation and the presented one is the following. Koch uses a special processing unit with many parallel processors, he mentions for QVGA (320×240 pixels) one processor per line memory unit. This means there are 64 Processing Elements (PEs). Although these processors are very small ones, this is an advantage at this point. All frequency analysis equations for one pixel are independent from the other pixel's results, so they can be solved at the same time. In opposition to this, my single threaded application has to calculate everything in sequence. As a consequence Koch's computation is done faster. An approach to use parallelization on consumer hardware is also part of this work and explained in Section 5.11.2.

4.4 Pose Finding

4.4.1 Aliasing

When sampling is performed, a continuous signal in wave form is split into equidistant values over time. The distance between those samples defines the resolution or accuracy by which the wave form is approximated. As a low resolution will always contain less information than a higher one, certain finer structures of a signal might get lost. Indeed, sampling wave forms can yield even worse effects described as *aliasing*. Aliasing occurs if a frequency f_{alias} is sampled at a rate f_s being less or equal than $2 \cdot f_{\text{alias}}$.

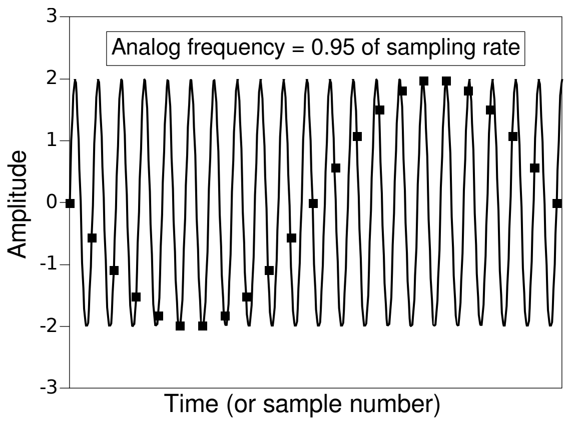


Figure 4.9: Aliasing causes a sub-sampled high frequency to look like a much lower one [Smith, 1997, p. 190]

A look to Figure 4.9 shows the effect. The frequency of the signal is at 95% of f_s. Combining the sampling points to a signal shows that they seem to be settled on a much lower frequency while they in fact represent samples from the narrow frequency painted in the background. In contrary, looking back to Figure 3.1 shows that a signal of 9% from f_s will be captured correctly. The observed effect is expressed by the *sampling theorem* in Equation (4.6) [Shannon, 1949]. It states that for correct sampling, the sampling frequency has to be more than twice as high than the highest possible frequency f_{max} in the signal. The first frequency causing aliasing in a system is the *Nyquist-frequency* $f_{\text{Nyquist}} = \frac{f_s}{2}$.

$$f_s > 2\, f_{\text{max}} \tag{4.6}$$

For the current system this means that there is an absolute limit for the frequency of an LED. Additionally, even LED-frequencies close to the Nyquist-frequency will cause aliasing. The reason therefore are the harmonics of the rectangular function seen in 8(a). While they will interfere with the frequency in search, this also holds a chance. The characteristics of a certain amount of aliasing from an LED may be used to detect and separate an LED more robustly from the background. This was not part of my work, but can be used with Koch's approach, which is described in the following section.

4.4.2 Selection of LED Frequencies

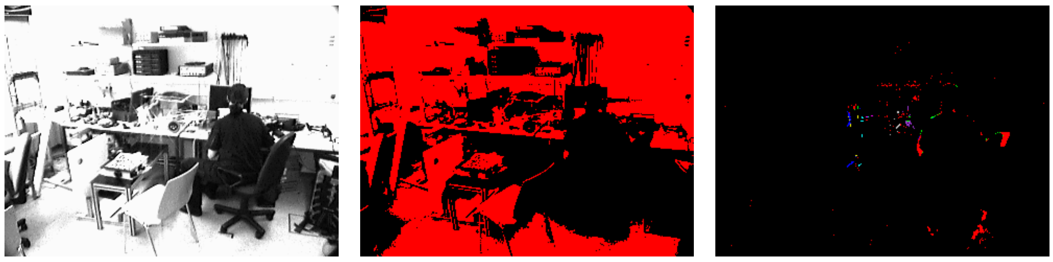
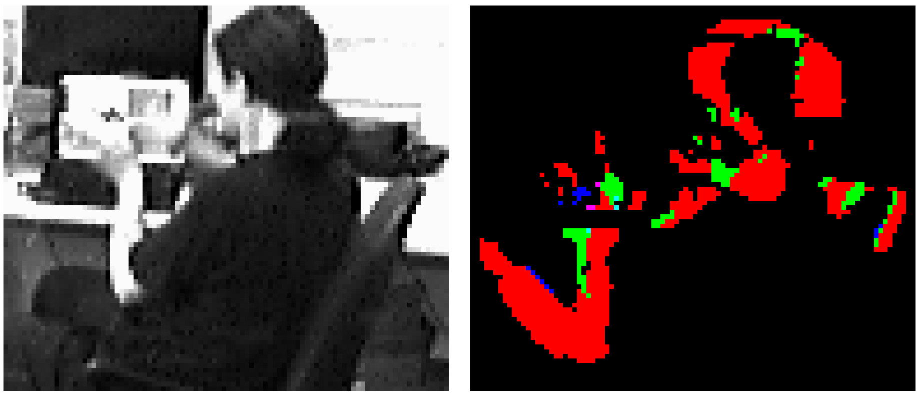

(a) Input Frame 5 (b) DFT Frame 5 (c) DFT Frame 26

Figure 4.10: The picture in the middle shows a visualized DFT-Segmentation after 5 frames of input Samples of the size 320×240 pixels. All red pixels are segmented as belonging to the LED with 1 Hz. After 20 frames many false positives are sorted out, as can be seen on the right. The picture on the left shows the real scene in frame 4. The DFT used for this image was 30 bins wide. See archive dft_1hz_false_positives in Section A.2 for details.

As described in Section 4.3.4 and 4.4.1, it is necessary to select the blinking frequency of an LED in a special range. In addition to the limit of the Nyquist frequency, it is necessary to avoid very low frequencies below 7 Hz. This is indebted by the general noise of the camera perception and the fact that smaller movements or for example the rapid movement of an arm in a scene can create frequencies up to 7 Hz like can be seen in Figure 4.11. The colors in those figures represent the affiliation of this pixel to a certain LED. Furthermore [Allen et al., 2001] mentions possible human motion frequencies of up to 20 Hz. Therefore they recommend capturing frequencies of 40 Hz and higher. However, my proposed system is limited to 30 Hz by the capturing rate of the camera.

Another reason for avoiding low frequencies is the fact that an LED, which blinks in an arbitrary frequency in the beginning will pretend to be another one. Without all the frames necessary to detect its particular frequency, it will show a signal close to 1 Hz signal due to the fact it changed at all. Later, after more information is put into the DFT input buffer, the wrong DFT-answers will vanish like it is shown in Figure 4.10.

In opposition to the tendency towards high frequency the initialization time has to be kept in mind. Choosing a higher frequency with alignment of one bin per LED causes the DFT resolution to increase. Following from that the number of samples needed to fill the whole set of input data raises proportionally. As the number of samples per second is fixed by a hardware limit



(a) Input Frame (b) DFT Frame

Figure 4.11: On the left, one may begin to fathom on a part of the input screen that I move my hands quite fast vertically. As a result of my body movement, there are false positives in the picture on the right. The magenta pixel in the center of frame 90 is the false positive value with the highest frequency I was able to simulate literally by hand. It corresponds to 6 Hz, as can be looked up in Section A.1. See archive dft_1hz_false_positives in A.2 for details.

this means an increasing time to initialize the pose estimation, although it is not compulsory to completely fill the input buffer before a pose can be calculated. Nevertheless, the results will not represent the exact frequency spectrum of the images observed.

Therefore I chose to use a compromise between high frequencies and a small initialization time. Starting at 7 Hz the frequencies are high enough to be distinguished from random noise. The bandwidth of 7 Hz up to Hz is split in steps of ≈ 0.43 Hz. By stopping at 14 Hz, neither the Nyquist-frequency is reached, nor the initialization will exceed ≈ 2.3 s with a resolution if $N = 70$. Those frequencies are created with a switching reaction time of 1 μs on the circuit controlling the LEDs.

4.4.3 Using the S-DFT

Performing pixelwise calculation of DFT-bins is computationally very expensive. For instance for a small QVGA image, which is quarter the size the camera used in this system provides, there are 76,800 Fourier spectra to calculate. At the desired capturing rate of 30 fps, for all those pixels the bin values have to be calculated below $33.\overline{3}$ ms. Looking at Equation 3.18, a naive implementation would perform at least N complex multiplications as well as the same amount of additions. It is necessary to minimize those calculations in a real-time environment, since the above mentioned calculation time is also used by other program components in a single threaded environment. Moreover the tests described in Section 5.11.1 revealed that the section computing the Fourier analysis is the bottleneck. In consequence a method to reduce the necessary calculations was used.

In general this is the application of the S-DFT described Section 3.1.3. It uses a special property of the DFT to calculate the spectral value $X_i(t)$ at time t by using $X_i(t-1)$. This results in a great increase of performance if a high DFT resolution is used, since there are only a few predefined frequencies we want to find in the image. In contrary, the FFT calculations of all bins for all pixels is wasteful, but also does not benefit from the previously calculated $X_i(t-1)$.

To further increase the benefit gained from this method it was implemented using Single Instruction Multiple Data (SIMD) instructions. The developed algorithm is described in more detail in Section 5.11.2.

4.4.4 RANSAC for the initial pose

After assigning an LED to each pixel inside the image, there has to be some means of grouping these pixels together. The pixels found for one LED may form a connected BLOB or multiple unconnected BLOBs of different sizes. Of course one of those BLOBs will represent the center of the corresponding LED, but the others may be unconnected parts of the center-BLOB or reflections of the LED's light on other objects. Those reflections will reveal similar values as the center as discussed in Figure 4.3.4. It is necessary to combine the information gained pixelwise into one pose per LED.

This is a critical step, as the quality of the initial pose rules how good any probabilistic method used later will perform. Therefore additional to the mentioned thresholding via normalization the segmented pixels left have to be analyzed. Valuable for this task is the knowledge of the fixed geometry of the object in search. It is known how the LEDs will be arranged on the image plane provided that a set of *extrinsic parameters* is known. Yet these parameters are unknown and the state space of a 3D$\mapsto$2D is large. As solution the knowledge of the segmented pixels LED affiliation is connected with the transformation parameters.

One method to reach this is RANSAC, described in Section 3.3. The implemented variant of the algorithm is visible in Algorithm 1cod:ransac. As a model, four planar points are used. Together with the intrinsic parameters calibrated in advance, a homography between the observed pattern with known geometry in *World* coordinates and its 2D representation on the image plane can be calculated. Using this the transformation *Hitch*[5] $\mapsto$ *FWCam*[5] [5] is evaluated for an arbitrary set of four image points with known world correspondence.

Hereafter it is necessary to rate the determined extrinsics. This is done by calculating the *back-projection* of each LED. In other words, the known world coordinates of each LED are projected to image coordinates using the calculated transform. This makes it possible to compare the calculated 3D$\mapsto$2D transform to the coordinates of the segmented pixels. In more detail each LEDs position is compared to each segmented pixel possibly representing this LED. If the Euclidean distance of a pixel under-runs the *point-threshold* it is added to the *consensus set*, the points supporting the hypothesis of this transform. The generation of a consensus set is shown in Algorithm 1cod:consensusSet.

After doing this for each pixel the set's size is compared to the *set-threshold*. Out of the models exceeding that threshold, the largest is remembered. In summary Figure 4.12(a) shows the

[5]See Table 8 for a list of coordinate systems

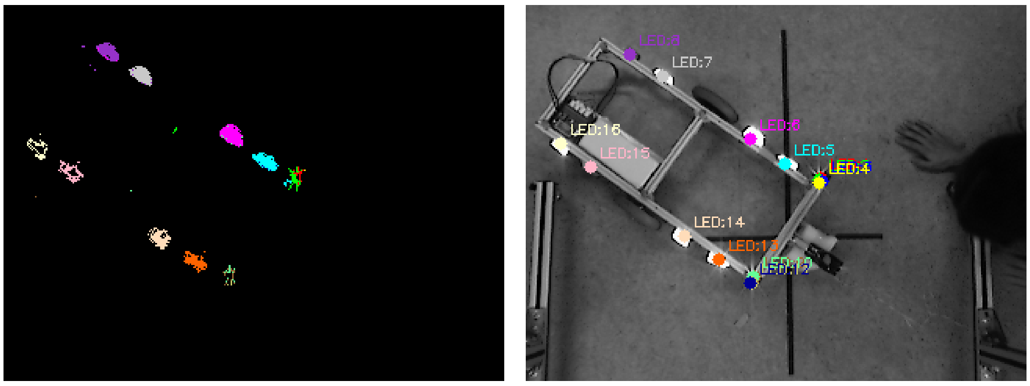
Algorithm 1 Generation of a transform from segmented pixels

```
Require: segmentedPixels
  for i = 0 to iterations do
    $sample \leftarrow$ drawSample();
    $tmpHitch2Cam \leftarrow$ calculateModel($sample$);
    $LEDProjections \leftarrow$ projectLEDs($tmpHitch2Cam$)
    $tmpSet \leftarrow$ generateConsensusSet($ledProjections, segmentedPixels$);
    if $tmpSet.size > setThreshold$ then
      $chosenSet \leftarrow tmpSet$;
      $Hitch2Cam \leftarrow tmpHitch2Cam$
    end if
  end for
  return $Hitch2Cam$;
```

Algorithm 2 generateConsensusSet()

```
Require: $segmentedPixels, LEDProjections$
  for all $pixels$ in $segmentedPixels$ do
    $euclideanDistance \leftarrow$ euclideanDistance(pixel, LEDProjections($pixel.LED$);
    if euclideanDistance $< pointThreshold$ then
      consensusSet.add($pixel$);
      consensusSet.weight += euclideanDistance;
    end if
  end for
  return consensusSet
```

processed frame after pixel segmentation, but before RANSAC was applied to generate a pose out of it. Using the described approach the system selects a model for the extrinsics, which produces the back-projected LED positions labeled in Figure 4.12(b).



(a) Segmented pixels (b) The back-projection of each LED's pose

Figure 4.12: Out of the set of segmented pixels a model is generated, which represents the extrinsic parameters

4.5 Movement of Objects

Core of the problem with frequencies in movement is that there is not the same kind of feature for an LED in every frame. A blinking LED may appear as dark pixel on one frame, but as a bright one on the next. This is why we cannot simply search for the LED in a single frame. Remember, to estimate the flashing pattern of an object, we need multiple images of it without much motion of the target. If, however, a moving, blinking object should be classified by frequency on an image, we have to know its position on each subsequent frame to compensate the displacement of those frames. The absence of this information is a problem. In fact, the trajectory of the found LED is the information we searched for, so the problem seems to be insolvable by frequency analysis. This is a key problem of optical tracking, which is visualized in Figure 4.13. As the motion gets faster, the tracking gets more and more difficult.

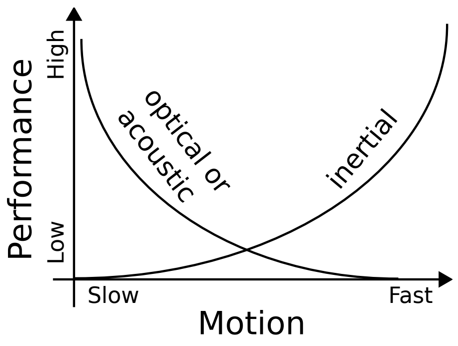


Figure 4.13: Performance of optical and inertia sensors [Allen et al., 2001, p. 57]

Therefore probabilistic estimation is used here. With this we simply guess a possible movement of the cart. To do this the state of the cart as 6D pose is captured in a state C_t, which denotes the state of the cart at time t (Equation (4.7)).

$$C_t = \left\{ \vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \vec{\omega} = \begin{pmatrix} \alpha_{\text{roll}} \\ \beta_{\text{pitch}} \\ \gamma_{\text{yaw}} \end{pmatrix}, \vec{v_p} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \vec{v_\omega} = \begin{pmatrix} v_\alpha \\ v_\beta \\ v_\gamma \end{pmatrix} \right\} \tag{4.7}$$

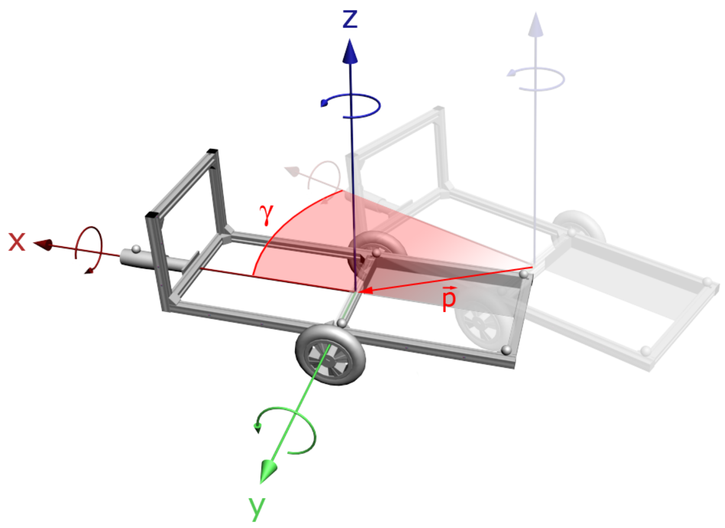


Figure 4.14: Cart coordinate-system *Hitch* in motion

It contains the translation $\vec{p}$ in mm and the fixed object velocity $\vec{v_p}$. The rotation $\vec{\omega}$ is represented as *Rodrigues Vector*, the same representation, which also thea ngular velocity tensor $\vec{v_\omega}$ uses. In such a vector the rotation for all axes is applied simultaneously, like a real object would move in space. Using these coordinates, we can describe the cart's position in space as well as its motion, what helps to guess the future position. All particle coordinates are considered as relative coordinates to the position which was determined at the estimation's start. Hence if the initial particle describes the transform *Hitch* $\mapsto$ *FWCam*[6], the transform stored in the particles is *Axis*[6] $\mapsto$ *FWCam*[6], where *Axis* is simply the moved variant of *Hitch* after initialization. Looking at Figure 4.14, both *Hitch* and *Axis* are visible. *Hitch* is shaded in the background while *Axis* represents the clearly visible coordinate system. The motion performed by the object in this case is defined by the translation vector $\vec{p}$ and the rotation angle γ on the z-axis.

[6]See Table 8 for a list of coordinate systems

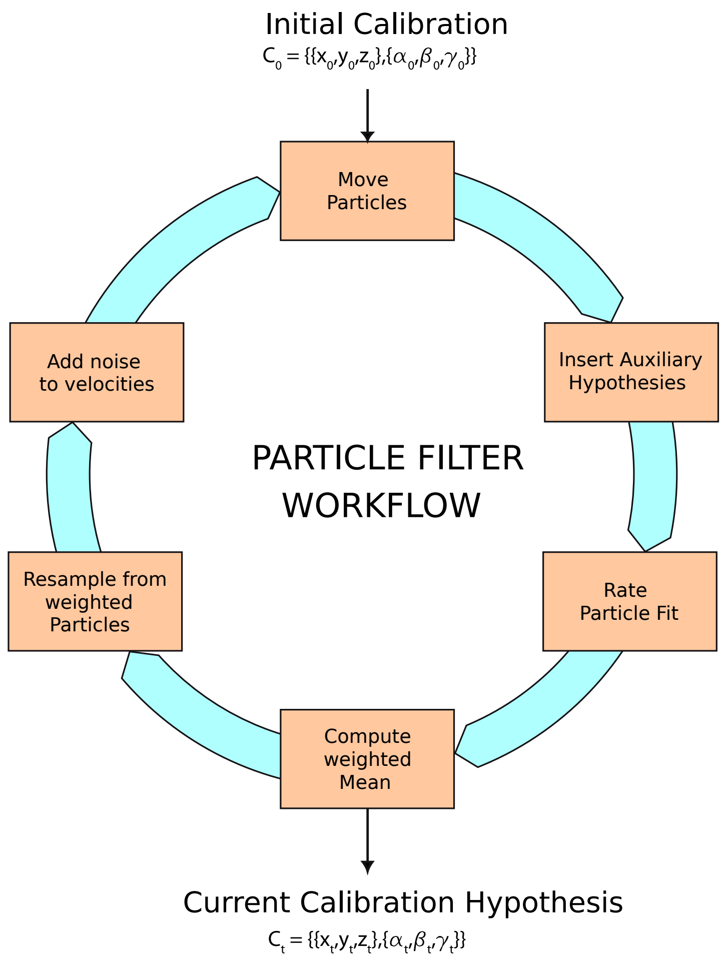

Initial Calibration
$C_0 = \{\{x_0, y_0, z_0\}, \{\alpha_0, \beta_0, \gamma_0\}\}$
Move Particles
Insert Auxiliary Hypothesies
Rate Particle Fit
Compute weighted Mean
Resample from weighted Particles
Add noise to velocities
PARTICLE FILTER WORKFLOW
Current Calibration Hypothesis
$C_t = \{\{x_t, y_t, z_t\}, \{\alpha_t, \beta_t, \gamma_t\}\}$

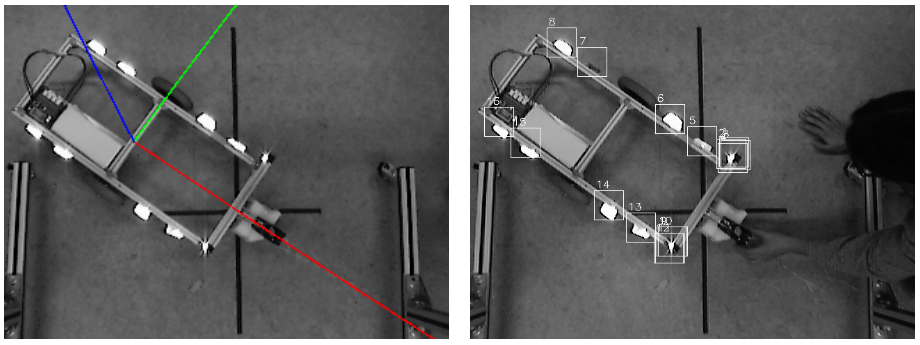
Figure 4.15: Workflow of the particle-filter

Schematically, the particle-filter tasks form a cycle denoted as a self-transition in Figure 4.1. This cycle is explained in more detail by Figure 4.15. Here we see an initial calibration C_0, which comes from the `PoseFinder`, which calculates a pose out of the images recorded from the stationary object. It contains all elements from Equation (4.7). As a result each run through the cycle produces a refined hypothesis C_t at time t.

Move Particles In this step, the particles are moved in the direction of their velocities v_p and v_ω state. By this, the particles represent different possible movements of the cart.

Insert Auxiliary Hypotheses Afterward some particles representing special behavior of the cart are added. They help to recover from estimation failures. In the proposed system at this step *standing*, and *random* particles are added. The latter simply help to find a nearby better solution while the former particles act as a substitute for a stationary cart. Their velocities are simply set to zero. If the cart has stopped, it is very likely that those particles provide a much better hypothesis than those still in motion.

Rate Particle Fit This is the core step of rating the quality of the particles. As the particle-filter itself cannot accomplish this, the help of the `Sampler` is needed again to check if the new position of a particle is reasonable provided the resulting frequency information gained by connecting it with the previous input data. Due to the interaction between the Fourier transform on the one hand, and the particle-filter on the other hand being one of the core features, it is explained further via Figure 4.18. To reduce the computational complexity of the Fourier transform again ROIs (Regions Of Interest) are defined. At the assumed LED-positions of the back-projections from a particle's calibration C_p, a ROI of configurable size is defined for each LED, like shown in Figure 4.16.



(a) *Hitch* coordinate-system (**X**,**Y**, **Z**) detected by the PoseFinder (b) The resulting ROIs

Figure 4.16: Mapping from a set of extrinsic parameters to the ROIs around the assumed LEDs

This projection is done for each particle in each step, while every time the ROI defines the area of pixels used as new input for the DFT. Resulting from this the ROI is moved in the image according to the changing extrinsics stored inside the particle.

While this alone is not sufficient to propagate the motion of the object, it opens up the possibility to use the frequency based searching again, because hypothetical motion performed by the cart is encapsulated in a particle moving in that particular direction. The pixel positions of the LEDs can be projected to the image plane using its transform. After doing this, the new intensity values are added to the history of that particle. Subsequently, the new signal influences the overall signal calculated by the frequency analysis of the pixels inside all LEDs' ROIs. That process is visualized by Figure 4.17. A path through this tree represents the history of a particle. Each level represents the total amount of pixels in the filter at that time. The number of pixels shown here is chosen for visualization purposes, normally the filter uses exactly the same amount of particles in each run. Resampling causes now a particle to be either dropped, like R_6, or to create children. In general the rectangles $R_{1...8}$ represent are images from one ROI, created to find a particular LED. If after the resampling step, a high amount of pixels segmented as the

LED in search is found in the ROI, the hypothesis of this particle is assumed to be good. If not, the particle will receive a low weight.

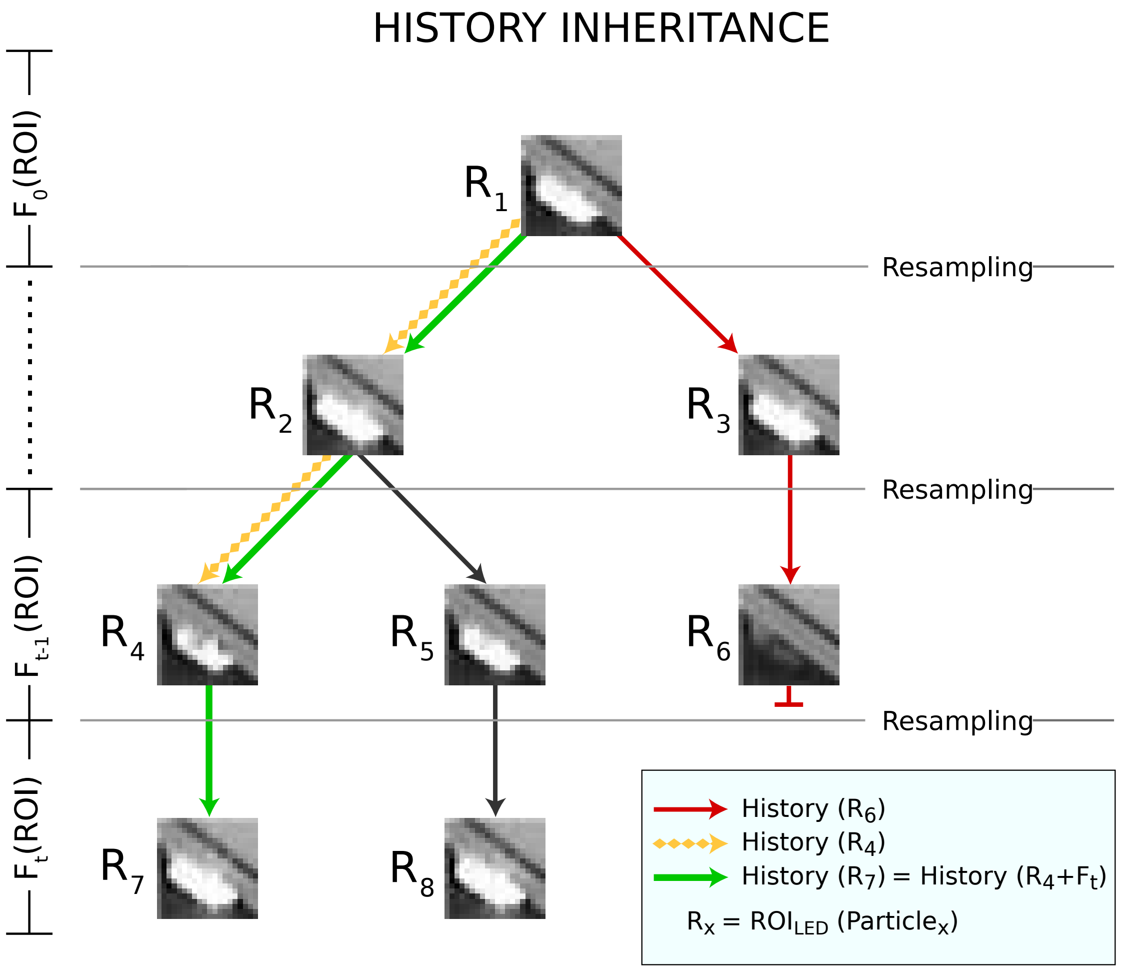


Figure 4.17: Sequence of ROI inputs

Compute weighted Mean By building a weighted mean using the weights gained in the last step the new hypothesis C_t for this loop-cycle is determined. This overall output of the particle-filter combines the information of all particles.

Resample from weighted Particles Dependent on their weight as defined in the step *"Rate Particle Fit"*, the particles will be either transferred to the new set of particles or dropped. In this context a transfer is realized as passing the circular input-buffer of all LED-ROIs. Through this, the particles representing wrong hypotheses will vanish while those providing a good estimate will stay in the focus of the algorithm. Furthermore, the resampled particles will inherit the input information gathered from their predecessor thus a new initialization is not necessary.

In the beginning the inheritance of the input history was left out and all particles had one combined history. This proved to be inappropriate as the information added by only one frame is rather small, compared to the history of N frames for an N-point DFT as a whole. Having separate histories for each particle preserves the whole sequence of a good particle and gives the possibility to add further frames in later filter steps by inherited particles.

Add noise to velocities The cycle is closed by the random element of the filter. At this step Gaussian noise, dependent on the time passed since the last time this was done, is added to each particle's velocities. The idea behind this is to add so much particles that one of them will eventually resemble the position of the cart after the real, unknown motion it performed.

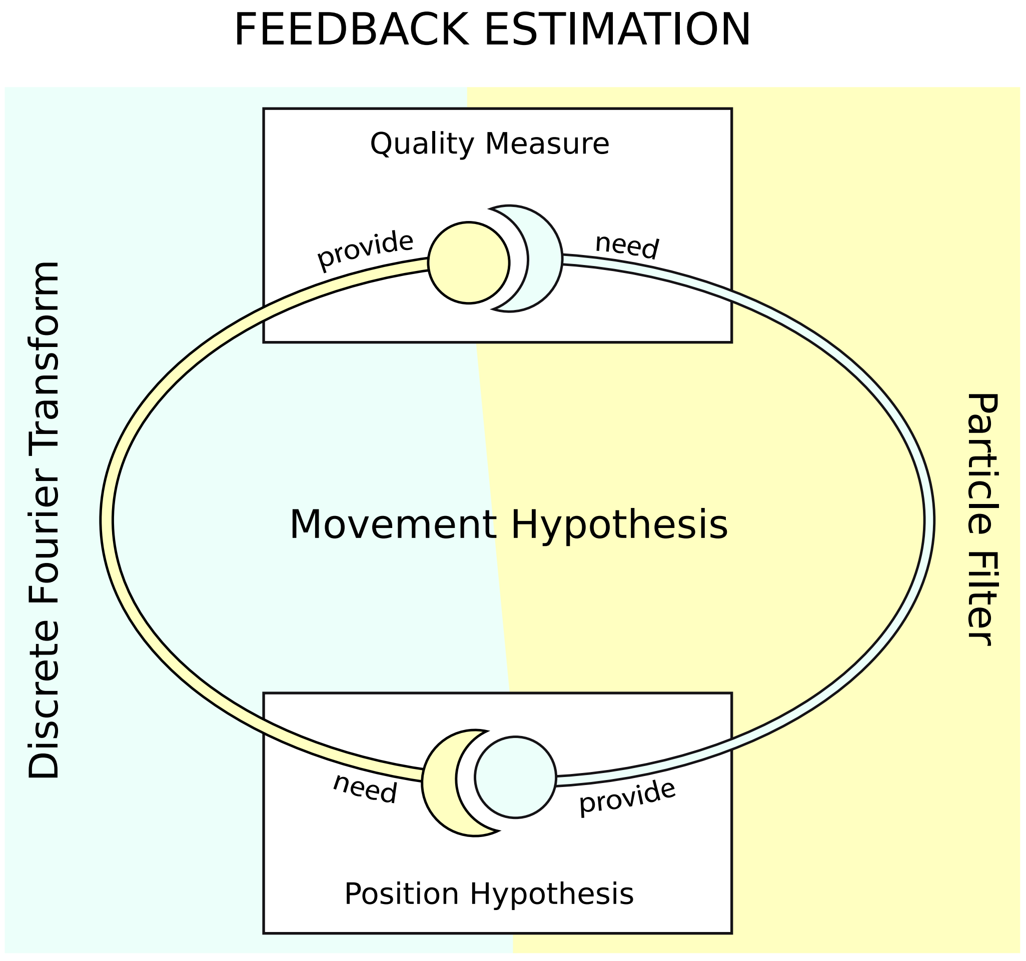


Figure 4.18: Feddback connection between the particle-filter and the DFT

To demonstrate one of the most noteworthy aspects of the presented tracking approach, Figure 4.18 connects the particle-filter and the DFT. On their own, both are insufficient to track a moving object. The frequency analysis on the one hand can only provide reasonable information about an LED's blinking if the trajectory of the object in the image is known. On the other hand, the particle-filter implements only a stochastic approach to handle hypotheses and needs other methods to judge the quality of randomly selected models. If, however, both are combined, it is possible to achieve a plausible trajectory of the object in search.

Based on this hypothesis the particle-filter estimates the next position of the cart and provides a new estimated position to the DFT. Using this position, it is possible to propagate the back-projected location of the pretended cart and its LEDs in the image. This is the information the DFT needs to add the right intensity values to the particular LED's input buffer. As a result, the DFT can offer the particle-filter a measure of how plausible this movement step was. By doing this for all possible particles, the ones next to the real LED positions get better weights than those moving randomly in the wrong direction. At this point, the particle-filter can again move its particles and the process starts again. In each step of this feedback, the hypotheses of the randomly spread particles get better, the filter *converges* until the result finally is a good *Movement Hypothesis*. Actually, the particle-filter will generate this hypothesis in each step, but the effect of the feedback of both components causes the hypothesis to be valuable at all. To start the feedback loop, an initial pose has to be estimated with a stationary object (see Section 4.4.4). Using the current frequency setup, this initialization takes $\approx$2 s (see Figure 4.4.2).

4.6 Physical Setup

The proposed system assumes a specific hardware setup. One of the assumptions is that the camera is fixed and does not move after calibration. This restriction is made to reduce the complexity of the system. While movement of the camera itself can be incorporated in the system, it requires knowledge about the moved distance. This could be determined, for example with an inertia sensor or odometry data from a pan/tilt unit, but is outside the scope of this work. Combining the information of both was for instance done by [Birbach et al., 2008]. According to the sensor properties already presented in Figure 4.13 this combination can compensate the weaknesses of both sensor properties.

The LEDs The used light sources can be seen in Figure 4.19. As described in the beginning of Section 2.2, most other tracking solutions rely on Infrared-LEDs (ILEDs). This circumvents the visibility of the light needed to detect markers in the scene. More common are indeed passive markers, which have the advantage of not requiring any circuitry for the markers, although for many passive systems there are also active ILED-Markers available to increase the operational range. Industrial standard approaches for tracking mostly rely on passive, retro-reflective markers, which are illuminated from the cameras direction in the infrared light spectrum. Camera devices illuminating passive LED markers are usually high priced special equipment. [Xsens, 2009] Furthermore, passive markers cannot directly solve the correspondence problem, meaning to distinguish the markers from each other, without the use of multiple perspectives from more than one camera. Instead, they have to form certain unambiguously rotatable compounds. For complex objects, this can require subsequent processing of the captured data by special software [Furniss, 2000]. This is due to the fact that all markers will reflect the light in a uniform way, as simply glowing LEDs would do. However, active ILEDs can be used for the proposed approach.

All in all the reason for not using infrared LEDs as markers for my thesis were medical concerns as the light is not visible by the human eye. Its natural closing reflex consequently will not work. The hardware for my thesis, which was preset by the DFKI[7] Bremen was therefore constructed with usual white LEDs. They are high power LEDs, which operate at 350 mA in the used circuit. At a viewing angle of 120°, they emit at 180 lm white light with a color temperature of 6500 K, which is similar to daylight.[8] Each LED can be controlled via an independent output of the driver board. Power is provided via a battery containing of 18 x NiMH cells connected in series having an overall capacity of 3000 mAh. Though the electronic hardware[9] was designed and built by the DFKI-Bremen electronic support staff, the program running on the microprocessor was written along this thesis.

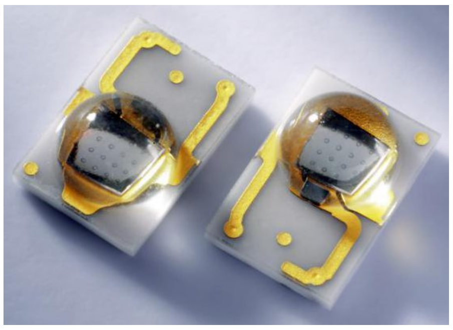


Figure 4.19: Used LEDs in a close-up shot.[10]

Cart While the fixed camera makes the sequential data connectable, the observed object, in this case the LED-pattern, is moving within the scene. Derived from this setup the pattern may be put on an arbitrary object which supports planar arrangement of the LEDs. For testing purposes, this setup was realized using an aluminum cart, which is visible in Figure Figure **??**(b). The cart was built for the case where a legged robot should find its way back to this cart. On all sides, there are LEDs embedded into the cart's aluminum bars. The LEDs were arranged in order to support the usage of the Zhang method described in Section 5.9. LEDs were placed on both sides and the back of the cart, so that at least four planar markers should be visible from most perspectives around the cart. Because the LED driver circuit supports only 16 LEDs, there are no LEDs on the top, and because the front of the cart will be occluded by the trekking vehicle there are neither LEDs on the front of the cart. The chosen arrangement on the side, with six LEDs in L-shape, was intended to add two additional feature points to the four necessary planar

[7]"Deutsches Forschungszentrum für Künstliche Intelligenz"

[8]LUXEON Rebel LXML PWC1 0100 Date: 02-16-2010

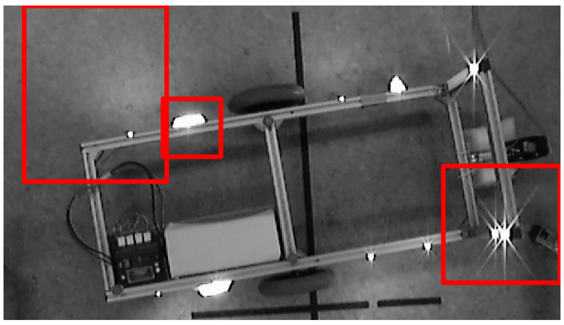
[9]See Section B.3for a diagram

[10]Image source: `http://www.leds.de/index.php?cl=media&mid=33845f51e51b840ea44fc333ece9d013-1&type=zoom`

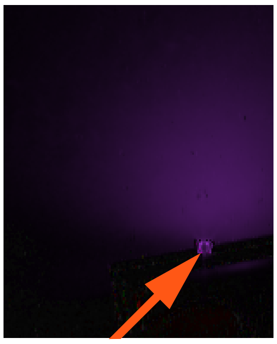
points of the pattern. As a result, the difference of using only four feature points in comparison to the pattern with six points should be evaluated. However, it turned out during the final tests that the L-shape is generally incompatible with the used algorithms. In [Zeng et al., 2008] it is mentioned that no three points of a four-point-pattern are allowed to be collinear for the calculation of a 3D space to image homography. Resulting from this, the only valid combination of LEDs on the L-shape is using two LEDs from one bar and two from the other one.

No three collinear LEDs To circumvent the problems of the L-shaped LED arrangement, which was determined to be unfeasible during the finalization, a top view of the cart was used. This way LEDs 5-8 and 13-16 are arranged in a rectangular pattern of eight LEDs which allows the application of the introduced algorithms. At this juncture, the problem of widely spread reflection compared to a very small range of a valid LED position occurred. To explain the problem, Figure 4.20 shows three different modes of an LED appearing in the image. Figure Figure 4.20(a) shows a complete overview from the original camera frame with frames around the regions shown below. The light emitted by the LEDs was white, so the color shown is originated by the system indicating the particular blinking frequency of this LED with purple. To visualize the intensity of the frequency signal a linear scaling of the color's values was done relative to the pixel exhibiting the strongest signal. The orange arrows were inserted after the processing to highlight the position of the LED.

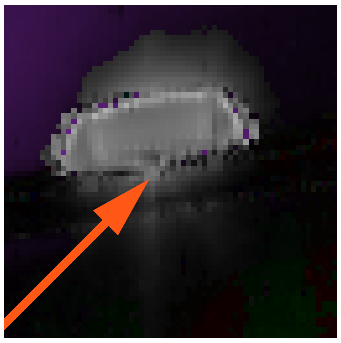
Beginning with Figure 4.20(b), an LED is visible, which was captured from a bird's eye view. It is visible that the LED itself is comparatively small in contrast to the light cone in the rest of the image, which is caused by reflection on the floor. Hence, a threshold to separate the LED's center from the reflection will work, but will result in a very small amount of pixels around the LED with a high risk of loosing the signal at all. Lowering the threshold will collect a usable set of pixels, but the general problem here is the position of the LED center relative to the gained set of pixels. The arrangement is not symmetric and it is not possible to quickly determine the origin of the light robustly. The basic concept of recognizing an LED in this system is the symmetric formation of pixels around a light source. The reason for this is to damp down the blooming effect (see Section 4.3.2) which can be observed looking at Figure 4.20(d). As the LED was directly visible by the camera this effect results in a symmetric spread of the signal around the LED's center. In consideration of this problem a funnel, open only to the top where the camera is placed, was placed below the LED. The resulting frequency signals for the gray colored LED are visible in Figure 4.20(c). All light rays, which would normally cause the reflections on the floor, as discussed before, are reflected by the funnel's walls. As a result, the shape of the funnel is observed on the frequency image. This allows to define the funnel's center as the core of the LED in search.



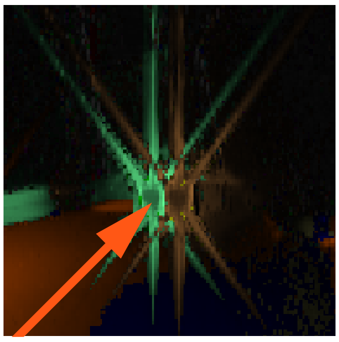
(a) Overview with marked regions



(b) Top view of an LED

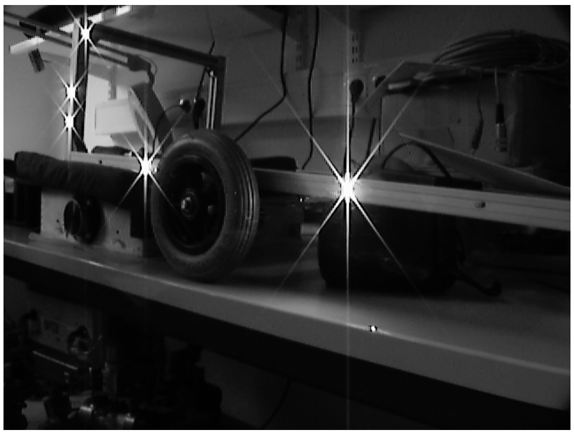


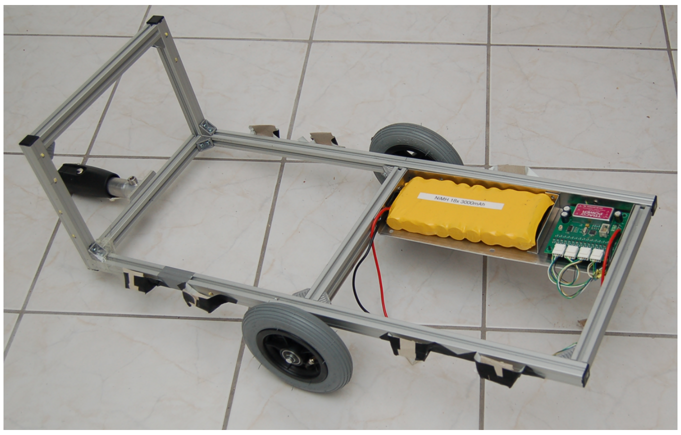
(c) Top view with light-funnel



(d) LED in front view

Figure 4.20: The figure shows the different relations of the observed light cone and the corresponding LED-position. For location of figures 20(b)-20(d), an overview of the original image with frames is given by 20(a). The center of an LED is marked by an orange arrow on each scaled sub-region. Other colors are generated according to an LED-affiliation and signal intensity (see Section A.1 for a color legend).


(a) The cart like it is seen by the system's camera


(b) Manually taken picture of the cart

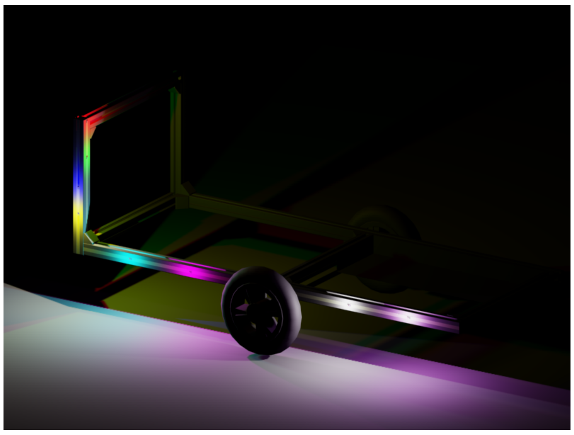

(c) 3D Model of the cart with LEDs marked as light spots in colors according to Table A.1

Figure 4.21: The used cart is shown in a photo, through the used camera, and as a 3D model. The colors in (c) correspond to those used by the system (see Table A.1). Dimensions can be looked up in Section B.1

.

5 Software

This section covers the implemented system to solve the given problem. Beginning with a short description of the used working environment I will describe the important parts of the software in more detail. It is divided into several modules, which implement the methods described in Section 3. All of them are compiled together to the `libtrack`, a `C++` programming language (`C++`) shared library which is the overall result of my software development for this thesis. For further implementation details see the reference manual `documentation.pdf` or the HTML version in `documentation/index.html` generated via Doxygen[11]. This document is created by extraction of code comments, which explain the implementation with more respect to the software architectural perspective and provide direct linked access to the library API. To make the relations between the described Modules more understandable a global illustration of the interconnections is provided in Figure 4.1.

All functions mentioned, variables, or preprocessor macros in the texts of a certain class refer to members of this particular class unless specified by explicit scope operators.

The system overview on page 23 explains the connections between the modules of the whole system. Starting on the very base each module performs a certain task and is interchangeable if the API is preserved. To keep it simple, this API is only constructed around one method, the `run` method. By its arguments it takes input and returns a preset output format. Herefrom arises the division of the modules into three steps, the *Acquisition*, *Segmentation* and *Pose Estimation*. Each of those steps requires a different API for the `run` method. Furthermore every module has a `Mode` object in common. Via inheritance from the interface class `interface/Module.h`, every module is able to receive a set of properties. Those are stored in the `mode` object. It is a class constructed to store certain properties like paths or a state configuration accessed during runtime. In order to pass or manipulate those values in tune with the `C++` initializer lists, all parameters can be read or written via method chaining.

5.1 Working Environment

I chose `C++` as programming language for development. First I decided to use OpenCV as core component for the necessary matrix algebra. By virtue of several issues with the FireWire interface of OpenCV I switched to FireWire library [Douxchamps et al., 2009] (libdc1394v2), a common Linux FireWire Application Programming Interface (API). The whole system is implemented as `C` programming language (`C`)/`C++` library with consistent interfaces. This approach benefits from a wide variety of available libraries.

To keep the development as easy and smooth as possible, no embedded systems were used directly on a robot. Instead a standard end user PC with common microprocessor architecture is used. For the purpose of simple development the system is single threaded.

[11]See [van Heesch, 2009]

[11]`http://www.archlinux.org`

[12]See [Sony, 2001] for technical details.

Table 5.1: Properties of the computer system used for development

Processor:	Intel Pentium M $2\times2{,}2$ GHz
Target Architecture:	i686-pc-linux-gnu x32
Main Memory:	2 GiB
Operating system:	Arch Linux @2.6.32 (IEEE 1394 module stack)[11]
Camera:	Sony DFW-500[12] on IEEE 1394

The camera model mentioned in Table 5.1 was built for industrial image processing and therefore provides better optics and can be controlled by standard IEEE 1394 imaging libraries. It operates at a maximum sample rate of 30 fps providing an 8 bit grayscale image.

It is necessary to keep in mind that the actual system, when moved to an on board computer, might operate on weaker hardware and another Linux distribution or even operating system. Therefore I have focused on the system performance and attempted to make the software as portable as possible. I used a general module structure throughout my classes, variable type macros and watched for warning free code. At last the whole library was furnished with a build system. This provides a convenient method for compiling the whole system on an arbitrary platform and checks for library preconditions. Due to this efforts it is no struggle to integrate the finished system on a mobile device.

5.2 Tools used

Table 5.2: Used Tools

Name	Version	Author	Description
OpenCV	1.1.0	[Sourceforge, 2008]	Open Source computer vision library originally developed by Intel. It implements a method to calculate the intrinsic, as well as the extrinsic parameters of a camera scene (see 3.2). It was also used for common matrix mathematical tasks.
g++ / GCC	4.4.3	[Free Software Foundation, 2008]	`C++` compiler from the GNU Compiler Collection (GCC)
GSL	1.13	[Galassi et al., 2009]	The GNU Scientific Library (GSL) was used to simulate random data for Gaussian distributions.
libdc1394v2	2.1.0	[Douxchamps et al., 2009]	This library was used as interface to grab frames from the FireWire device.
Boost	1.41.0	[Dawes et al., 2009]	The library `boost::filesystem` is used as platform independent file system API, `boost::regex` helps sorting recorded data while `boost::test` is used to perform unit tests on the library.
CMake	2.8.0	[Martin and Hoffman, 2008]	Platform independent build-system.
Doxygen	1.6.2	[van Heesch, 2009]	Used to compile a reference manual out of the comments in the source code.

5.3 Modules

The developed software is split into modules as seen in Figure 4.1. As an example application with access to the developed library the program named `watch` is provided. This command-line tool is able to use all developed modules, switch them on and off, and to pass parameters to them. Listing 1 shows the parameters to use the modules.

Listing 1: Module arguments for `watch` application

```
Global Thesis Tool (watch - Ruben Stein 2009/2010)
This tool utilizes the features implemented in the libtrack for my diploma thesis.:

General:
  -h [ --help ]         Produce this help message

Modules:
  -c [ --capture ]          Grab frames from camera with provided index.
  -e [ --differ ]           Calculate 16 bit difference image between current
                            and last frame.
  -d [ --dft-segment ]      Use Fourier transform to segment image.
  -l [ --load ] arg         Load frames from provided path and filename-pattern
                            (e.g. --load data/img*.ppm).
  -f [ --particle-filter ]  After initialization (filling the S-DFT), use
                            a particle-filter to estimate further movement of
                            LEDs.
  -p [ --pose-finder ]      Use RANSAC and Zhang's algorithm to find the best
                            solution to the perspective-N-Point-problem.
```

In addition to the plain functionality to use the implemented algorithms, the application provides also some convenience functions. Amongst them are routines to save all kinds of calculated results with and without debug drawings, plotting of resulting coordinates and custom termination conditions. These are listed in Listing 2

Listing 2: Convenience function arguments for `watch` application

```
Interactive:
  -H [ --halted ] [=arg(=0)] program will be started halted, press p to
                             continue or n for next frame
  -T [ --halt-term ]         Instead of waiting for user action, terminate
                             after the amount of frames specified via --halted.
  -S [ --save ] arg          Save images to this folder using their predefined
                             subfolders.
  -q [ --quiet ]             Mute standard text output. This does not affect
                             plotting output.
  -O [ --out-original ]      Show the original input frame.
  -M [ --out-sample ]        Show the downsampled version of the input frame.
  -D [ --out-dft ]           Generate a visible representation of the
                             segmentation result from the Sampler.
  -F [ --out-pose ]          Highlight each found LED of the final estimation
                             with a colored circle corresponding the defined
                             LED-colors.
  -J [ --plot-led-poses ]    Prints the coordinate for each LED in 3D
                             coordinates of the camera coordinate system. The
                             output is not formatted to be interpreted by
                             plotting programs.
```

The supported debug-drawings are listed in Listing 6. All of them will be printed on every used image (e.g. original frame, down-sampled frame, segmentation result) and are dynamically scaled. Clicking on one of them during runtime will show details about the target pixel. If the target was located on a pixel inside a DFT-ROI, the internal data such as time-domain signals, frequency-domain output and normalization result will be printed to the console.

Listing 3: Possible debug-drawing arguments for `watch` application

```
Graphics:
  -A [ --relative ]         Each DFTSampler draws color values relative to its
                            maximum.
  -W [ --windows ]          Use OpenCV to show images.
  -C [ --draw-cart-axes ]   Draws the coordinate system on the cart (X(red)
                            Y(green), Z(blue).
  -E [ --draw-difference ]  Paint difference image between current and last
                            frame.
  -L [ --draw-labels ]      Paints label next to found LEDs.
  -Z [ --draw-particles ]   Paint one pixel per particle's estimated LED
                            position in the particular color of that LED.
  -P [ --draw-pose ]        Highlight each found LED of the final estimation
                            with a colored circle corresponding the defined
                            LED-colors.
  -R [ --draw-rois ]        Paint current ROIs from sampler.
```

5.4 StateFinder

The `StateFinder` implements the HMM stack inside the software. Important to mention at this point is the difference between the development status of HMM- and Fourier stack. While the HMM stack was implemented until the concept worked for small 32×32 images, the Fourier stack was continued as productive code.

Beginning with the required input for the `StateFinder`, the `HMM` class comes into focus. This class implements the core concepts of a Hidden Markov Model like described in 3.1.1. To reduce errors by linearly repeating code, the concepts of the described tripartite state model are encapsulated in generation functions. In order to allow the automatic generation of a HMM's states are represented as raw numbers hidden by a naming pattern, which describes their task.

- `L1_0_UP_0_ON_POS_FUL`
  LED index
- `L1_0_UP_0_ON_POS_FUL`
  Phase of the LED, meaning the horizontal state index (cf. Figure 4.7)
- `L1_0_UP_0_ON_POS_FUL`
  The current edge (up, down, zero) of the LED from list {`UP`, `DO`, `ZE`}. Depending on the previous intermediate-state it either rises, falls or does not change at all.
- `L1_0_UP_ON_POS_FUL`
  Will take values from {`ON`, `OF`} controlled by the LED being switched *on* or *off* in this state.

- `L1_0_UP_ON_POS_FUL`
  Indicates if a $\Delta\,\iota$ signal is positive, negative or zero. $\{$`POS`, `NEG`, `ZER`$\}$
- `L1_0_UP_ON_POS_FUL`
  Divides the strength of the $\Delta\,\iota$ signal into two classes according to the blue medium and full size bars in Figure 4.6 $\{$`FUL`, `MID`$\}$

Another property which can be evaluated at this point is the period or state of the LEDs. As the LEDs are synchronized among each other, a wrong period in an LED may give a clue for an outlier who is not based on the LED pattern on the robot.

Based on the path calculated by the Viterbi Algorithm (see Figure 4.3.3) it is possible to determine which LED is represented by the input. Therefore the output of this module contains the most probable meta-state for each pixel, what corresponds to the LED it belongs to, and a probability value for this hypothesis.

5.5 DFTSampler

This class implements a pixelwise sliding window discrete Fourier transform. This means for every pixel, a frequency spectrum is kept. In order to calculate it, the sequence of 8 bit values for a pixel with coordinates $[x, y]$ over several frames is interpreted as discrete time-domain signal (see Figure 4.2). The frequency spectrum tells which frequencies are contained in the last N values for this pixel. Based on this spectrum a certain score for each LED is calculated (see Figure 4.3.4). In the end this states which LED's frequency is most likely to get sampled by this pixel's grayscale values.

The common work-flow for this class starts with pushing the input signal to the buffer via `putInBuffer`. After this a call to `sdftStep` calculates the DFT-Data. In more detail it calculates for every pixel a set of frequency values for arbitrary bins defined in `Mode::calcBins`. Each of this bins is mapped to a particular frequency by `Mode::binToLED`. In other words if the normalized value for this bin is the highest in one calculation step, the pixel gets its LED affiliation by the mapping of the bin-index to an LED via `Mode::binToLED`.

Internally this class has the task to compute many similar steps for a large amount of items, in this case pixels. This property yields the possibility to parallelize this task. Many independent *workers* can calculate the values for each pixel without needing each other's results. In Contrast to Koch's approach (see 4.3.5) I used a method for small-scale parallelization on consumer hardware, which is described in more detail in Section 5.11.2.

For the pose calculation only the segmented pixels are needed (`pixels`). For the construction of a visible representation of the data on the other hand also not segmented pixels have to be stored (`pixelsImgOrder`). Those values can be parsed sequentially to form a colored image out of the DFT-Data (`getPaintableDFTImg`). Table A.1 explains which color is used for which LED and also explains the special *relative* image. The mentioned LED-indices can be looked up from the technical drawing in Section B.1.

5.6 Sampler

This module can be looked upon as infrastructure unit above the DFTSampler. In terms of the API, it behaves exactly like the DFTSampler. It is defined for a certain image size and has the task to provide an LED index for each pixel, based on Fourier analysis of subsequent frames. In fact every DFTSampler in a system can be replaced by the Sampler. However this relation is not always reversible. The reason is that the sampler is a skeleton which allows to define multiple DFTSampler instances on one image.

Starting with a raw Sampler there will be no segmentation at all. In order to start the process one has to define a certain Region in the sampler's range. This region is henceforth represented as a DFTSampler instance. Internally this DFTSampler does not know about its limitation to a specific image area, it operates on a normal image. The key difference is that the Region structure passes the Sampler's image to the DFTSampler. In doing so it is possible for multiple DFTSamplers to work on one bigger image without the need to calculate the whole image. So the Sampler allows for ROIs inside the image, which get segmented by the Fourier stack. Through this a huge amount of calculation capacity can be saved.

Being able to selectively calculate over smaller regions reduces the computational complexity of the pixel segmentation. This is because the sampler is able to reduce the resolution of an input frame and calculate the Fourier information for the small image. Although the accuracy is reduced as several pixels information are combined, one can now define a new DFTSampler to calculate the Fourier information for a small extract of the higher resolution image. Because of this the performance of the overall system is significantly improved while preserving the accuracy of the initial input-frame resolution.

Listing 4: `Sampler` arguments for `watch` application

```
Fourier Sampler:
  -w [ --normalization-threshold ] arg (=0.058)  Threshold for the normalized
                                                 values of the Fourier
                                                 Transform. If a pixels value
                                                 for the highest bin exceeds
                                                 this value, it is accepted as
                                                 segmented for an LED.
  -y [ --pf-normalization-threshold ] arg (=0.5) Same as above, but only
                                                 applies in ROIs of
                                                 ParticleFilter.
  -z [ --pf-normalization-target ] arg (=1)      Target value around which the
                                                 threshold is applied (only
                                                 for ROIs of Particle Filter)
  -o [ --normalization-target ] arg (=1)         Target value around which the
                                                 threshold is applied.
  -i [ --region-rect ] arg                       Sets the region in pixels for
                                                 which the DFT values will be
                                                 computed. The region must be
                                                 specified in sample
                                                 coordinates. Allowed argument
                                                 numbers are 2 (<x==y>
                                                 <width==height>, e.g. -i 4 10)
                                                 or 4 (<x> <y> <width>
                                                 <height>, e.g. -i 3 2 640
                                                 480).
  -k [ --samples ] arg (=30)                  Width of the Fourier comb, so
                                                 number of frames to use when
                                                 calculating the Fourier
                                                 values.
  -s [ --sample-size ] arg (=320 240)         Size in pixels to which the
                                                 original input image will be
                                                 scaled down. May be specified
                                                 as single value for a square
                                                 (e.g. -s 4) or rectangle
                                                 <width> <height> (e.g. -s 640
                                                 480).
```

5.7 ParticleFilter

The ParticleFilter was implemented to make tracking of moving objects possible. In contrast to the situation for a stationary object, an object in motion cannot provide the necessary data for a frequency analysis. As explained in Section 4.1, multiple input frames shall be combined to detect certain intensity patterns caused by blinking LEDs. Although observing a stationary object is possible because subsequent frames can be computed to a pixelwise frequency image, this is not possible for blinking LEDs which move in the image. For this to work, the system has to be aware of the motion the blinking object performs. Another approach would be to bypass the frequency analysis once the system is calibrated and knows an initial position of the LEDs in space. We could now switch all LEDs on and simply try to follow their intensity values in the image. However this was not recently implemented due to a lack of time.

In either case a state estimation approach based on a particle filter will be used. The idea behind its use is to treat the particles of the filter as random hypotheses of the object's movement. Once the PoseFinder calculated an initial transformation between world and image coordinates, the particle-filter is initialized. Its particles all contain a complete hypothesis about the relative pose of the camera with respect to the object.

Listing 5: `ParticleFilter` arguments for `watch` application

```
Particle Filter:
  -K [ --automatic-start ] [=arg(=1)] (=0)   Particle Filter starts after
                                             input-buffer for DFT is filled.
  -t [ --initial-noise-time ] arg (=1000)    Time in ms that should be used to
                                             mime an initial dynamic step to
                                             spread the particles after
                                             initialization.
  -n [ --particle-number ] arg (=40)         Number of particles used.
  -x [ --patch-size ] arg (=10 10)           Size in pixels for the region
                                             around an LED as center, for which
                                             the Fourier values are calculated.
                                             May be provided as single value
                                             for a square or two values for a
                                             rectangle.
  -r [ --sigma-rotation ] arg (=5,5,5)       Mean noise on the rotation axes in
                                             degrees per second. May be
                                             provided separate for each axis
                                             x/y/z or as single value separated
                                             by a comma (e.g. --sigma-rotation
                                             1,0.3,4).
  -j [ --sigma-translation ] arg (=10,10,10) Mean noise on x/y coordinates of
                                             particles in mm per second. May be
                                             provided separate for each axis
                                             x/y or as single value separated
                                             by a comma (e.g.
                                             --sigma-translation 1,0.3,4).
  -b [ --reinit-threshold ] arg (=1.E-3)     If the summed up weight of all
                                             pixels falls below this value, all
                                             pixels are reinitialized using the
                                             current hypothesis.
```

5.8 PoseFinder

This module receives a `samplerOutput_t`, a map of segmented `MotherPixel`s sorted by LED from a `Segmenter` module. Using RANSAC (see Section 3.3) it generates a `Calibration`, containing the extrinsic parameters of the assumed object-pose. Therefore from four different LEDs' pixels one pixel is chosen randomly, a model is calculated, and finally rated according RANSAC's algorithm. The process is repeated for a configured number of `iterations` and returns the best rated model generated along the way.

Listing 6: `PoseFinder` arguments for `watch` application

```
Pose Finder:
  -v [ --iterations ] arg (=80)     Number of iterations for the RANSAC algorithm
                                    in PoseFinder.
  -g [ --pointThreshold ] arg (=10) Maximal Euclidean distance in pixels from a
                                    point to the current model.
  -u [ --setThreshold ] arg (=10)   Minimum number of points supporting a model
                                    hypothesis (below pointThreshold) to get
                                    into the consensus set.
```

5.9 Camera calibration

Calibration is done using a set of OpenCV methods in a small tool shipped with the windows distribution of the library. It can be found in `src/calibration`. The theoretical background for this calibration can be found in 3.2. As input this program gets pictures of a chessboard with known geometry observed in at least two different attitudes. I used 12 valid, meaning fully recognized, different images of a 7×7 squares chessboard (counting only inner edges) whereas one square has the dimensions of $11.2\,cm$ edge length (Figure 5.1(b)).

Internally the methods used for the propagation of the needed camera parameters are derived from Zhang's method [Zhang, 2000], although the distortion parameters will be calculated via [Brown, 1971]. Zhang's method needs at least 4 planar points, arranged in known world coordinates and solved correspondence as input. Given the fact that the lens is of fixed focal length, the intrinsic parameters have to be calculated only once before the whole system can be used. However: Remind that the focal distance f_x, f_y as well as the principal point c_x and c_y need to be scaled proportionately to the image size used during calibration.

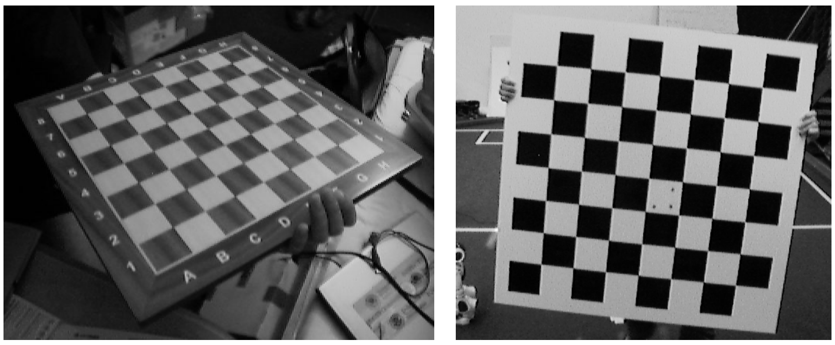

(a) Small Chessboard (b) Big Chessboard

Figure 5.1: The two chessboards used for calibration as frame from the camera

During runtime of the program, the intrinsic parameters are stored in memory. The extrinsic parameters have to be evaluated every time the relative poses of camera or LED-pattern towards each other have changed.

5.10 LED Driver Board

To generate a certain blinking frequency at the LED output, an interrupt was used. It is triggered at intervals of milliseconds defined by `IRQS_PER_SECOND`. In an Interrupt Request (IRQ) the kernel stops for a certain scheduled event. For most of the tests, a value of 1,000,000 IRQs per second was used, corresponding to a reaction time of $1\,\mu$s for switching an LED on and off. The micro-controller-code necessary to raise the interrupt and configure an according timer was adapted from [rn-wissen.de, 2008]. In every interrupt the controller checks if it is time to switch the output for an LED by comparing to a pre-calculated array fitting to the desired frequencies. The source code for this is located in `led-driver.c`.

Analogous to the previously described IRQ approach, the generation of blinking patterns for the HMM algorithms is built. Instead of specifying a frequency for each LED output, a binary pattern is specified. Together with a value of the capturing rate (`CAPTURE_RATE`) and the maximum signal length (`MAX_SIGNAL_LENGTH`) the patterns are stretched to let every *signal*, one bit of the binary LED pattern, last two frames of the capturing rate. This allows the camera to register a change in an LED's state at its particular sampling rate.[14]

[14]See Figure 4.3.3 for details on the chosen length of a signal

5.11 Runtime Optimization

5.11.1 Profiling

Before the optimization was done, I used a *profiler* to check which parts of the software actually slow down the process. Optimizing program code without use of such a tool is unprofitable, since only those regions consuming much time, really slow down the process. The used tool for this analysis was Valgrind [Seward et al., 2008]. It is a suite for debugging and profiling, containing the *callgrind* tool. This *profiler* simulates a virtual Central Processing Unit (CPU), and optionally also the behavior of caches. In order to connect a normal program with this tool, the program to profile is simply started as parameter. Valgrind now checks the debug output of the executable and lets tools like *cachegrind* insert special instructions for a later analysis. Not only the executable itself, but all calls to shared libraries are included in the analysis. In contrast to this, many other profilers are not capable of tracing shared library calls.

The generated output includes statistics about instruction fetches, cache misses, loop relations and the corresponding assembly instructions. Important in this context is: profiling code without compiler-optimization cannot represent the performance of the optimized program. It is possible that changes which increase the performance in unoptimized code actually decrease the performance of optimized code. One reason for such behavior includes cache misses, which often outweigh the *cost* (consumed time) of repeated instructions.

5.11.2 Data Representation and Streaming

Internally, the pixelwise information of the `DFTSampler`[15] is stored in long, connected chunks of memory. Through this a lot of optimization is reached by simply storing them in subsequent memory areas. The module supports caching facilities, and also eases fetching multiple values manually by the use of Single Instruction Multiple Data (SIMD) instructions.

The Fourier values are calculated using a *shifting property* of the complex Fourier calculation (see Section 3.1.3). This way only one complex multiplication and two real addition operations are necessary to calculate a new Fourier value for a single bin. For this to work certain buffered values must be available. Those are the Fourier value $X_k(n-1)$ in an N-Point DFT for the k^{th} bin at frame n - $X_k(n)$, as well as the input pixel intensity values v_{n-N} and v_n. Consequently for the input values there is a circular buffer to read the first and last input signal of the window. Furthermore this calculation was, among others, transformed to a look-up table calculated during the creation of the `DFTSampler` object. This allows values with a discrete value range to be directly connected to a calculation output without repetitions of calculating them in each instance.

SIMD instructions allow application of one operator to more than one value in only one CPU instruction, a process called *vector processing*. While vector processing was common in scientific

[15]See Section 5.5 for a description of the module or Section 4.3.4 for the concept

computation, it was integrated to the end-user market by Intel through Multi Media Extension (MMX) [Peleg and Weiser, 1996]. The new instruction set was inspired by the needs of common tasks in sound or video calculations, which needed to perform the same operation on a large amount of data. Currently almost every consumer level microprocessor architecture is able to execute a certain subset of SIMD instructions. One popular subset are the Streaming SIMD Extensions (SSE) instructions by Intel, which introduce eight 128 bit wide registers (`XMM 0-7`). They are aligned to the needs of common multimedia work-flow. Based on those registers the instruction set offers many basic calculations [Raman et al., 2000]. For my thesis I used the SSE3 instruction set, an addition to the SSE instruction set, that was published in 2004 [Intel, 2009b, 5.7 p.157].

Consider the following as an example of a use case that fits utilization of SIMD instructions, a standard application for streaming operations is the calculation of the maximum, considering two disjoint sets of numbers which are stored sequentially regarding their particular set. It can be calculated using the `__m128 _mm_max_ps(__m128 a, __m128 b)` instruction [Intel, 2007, p. 33]. In this instruction, `__m128` are simply 128 bit registers with 16 bit alignment, a data-type representing the internal registers. In Figure 5.2 a *packed instruction* (indicated in the above example by the suffix “`_ps`”) is shown. It connects the sets of values column-wise to calculate the corresponding column entry in the result-set. This example uses, like in this thesis, four 32 bit single precision values. Since the size of the registers is fixed, only two double precision values can be calculated by a packed instruction.

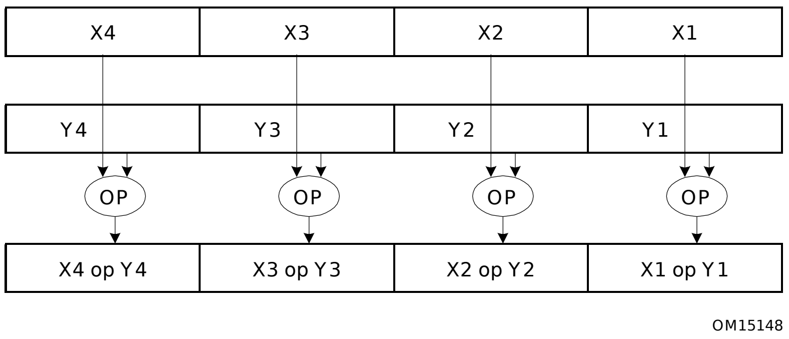


Figure 5.2: Packed instruction, which operates on two sets of four elements each, connecting the values column-wise to calculate a result also having four elements (Image Source: [Intel, 2009a, p. 89]).

The above mentioned instruction is part of the *intrinsic* `C` instructions of Intel. These `C`-Macros ease the use of SSE-Instructions by freeing the programmer from the need to directly use assembly code and thus from time intensive address management for registers. Instead a new data type `__m128` is introduced to represent the `XMM` registers making it easier to integrate the instructions to `C`/`C++`-Code.

Using these instructions in the core of my applications, increased the performance of the S-DFT calculation drastically. All internal calculations of the function `DFTSampler::sdftStep` were rewritten in a way to support streaming operations. Therefore the sizes of containers had to be adjusted for processing in 128 bit chunks, while the used data type remains variable so the actual implementation may support double precision as well, if this is needed.

The core sequence to compute the complex multiplication of a bin, which is the most costly operation in terms of CPU instructions, was adapted from [Smith et al., 2004, p. 23]. All the other auxiliary transformations towards the SSE version of `sdftStep` were developed within the scope of this thesis. Apart from the SSE mechanisms there is also much use of look-up tables for constant expressions like `DFTSampler::expPiLUT`, which holds all combinations of passed bin indices with pre-computed multiplications of 2, π, and N. In contrast the core instructions only deal with the unavoidable computations.

5.11.3 S-DFT-Algorithm

Listing 7: Naive code for S-DFT algorithm

```
for (size_t pixelIdx(0); pixelIdx < pixelIdxMax; ++pixelIdx)
{
  calc_t*     cSigSum     = sigSum          + pixelIdx;
  calc_t*     cSigSumQuad = sigSumQuad      + pixelIdx;
  calc_t*     cStdDev     = stdDev          + pixelIdx;
  complex_t*  out         = fourierOutput   + pixelIdx;
  calc_t*     norm        = normalised      + pixelIdx;
  calc_t *    polar       = polarOutput     + pixelIdx;

  //calculate the standard deviation
  *cStdDev = sqrt(   (       *cSigSumQuad /       N )
                   - ( quad(*cSigSum)     / quad(N))
                 );

  for (size_t binIdx(0); binIdx < bins; ++binIdx)
  {
    complex_t* cOut   (out    + binIdx);
    calc_t*    cPolar(polar + binIdx);
    calc_t*    cNorm  (norm   + binIdx);

    //subtract first signal in buffer, add the last
    *cOut -= firstIn - lastIn;

    //do complex multiplication
    *cOut *= exp(std::complex<comp_t>(0, 2 * PI * calcBins[
       binIdx] /N));

    //calculate norm of output
    *cPolar = sqrt(quad(cOut->real()) + quad(cOut->imag()));

    //normalise the result;
    *cNorm = (*cPolar * 2) / (*cStdDev * N);

    //select maximum bin
    if (*cNorm > normMax)
    {
      normMax = *cNorm;
    }
  }
}
```

The algorithm presented in Listing 7 shows a naive method for calculating the S-DFT. This implementation is used as comparison to the SSE-enhanced version. It iterates over all pixels and calculates the bins' values. In Line 11 the standard deviation is calculated using *computational formula for the variance*[16] to minimize computation. Afterward, in Line 15, the loop for computing each bin's value begins. Line 22 adds the current intensity value of the pixel ι, and subtracts the first one from its input buffer. Together with the complex multiplication in Line 25, these computations represent the S-DFT calculations from Equation (3.19). Subsequently the norm of the complex number is calculated in Line 28. Regarding SSE instructions it is necessary to mention, that up to this point each calculation is done twice, since two complex 32 bit values result in one 32 bit polar value. Finally the values are normalized in Line 31, according to Equation (4.4), and in Line 36 a norm value superseding the current maximum is saved.

It has to be mentioned that the implemented SSE algorithm was developed at a state where the results of multiple bins per Fourier Spectrum were used. In other words, the initial search for an LED requires every bin to be calculated, because we do not know which LED the pixel might represent. Later, if we are searching for a particular ROI, where only the bin representing the LED of that ROI is of interest, the algorithm is not optimal. The reason is that the current implementation includes one bin-wise vectorized loop. That means the results are calculated for four subsequent bins. If now only one bin has to be calculated, the other three computations are wasted. As a result, there is still a lot of potential for performance increases if the algorithm is implemented to calculate four subsequent **pixels**'s values simultaneously instead of **bins**'.

5.12 Performance

To show the performance of the developed algorithm for DFT computation three versions of solving the calculations were compared. First, the FFTW-library [Frigo and Johnson, 2005] was used as performance indicator. This library is a state-of-the-art Fourier transform library. Second, the SSE-version of my S-DFT implementation was used. Additionally the naive implementation of the S-DFT was included to show the performance boost of the SSE version in comparison. Figure 5.3 shows the compared runtimes for 100 iterations of a 640×480 input frame.

[16]German: "Verschiebungssatz"

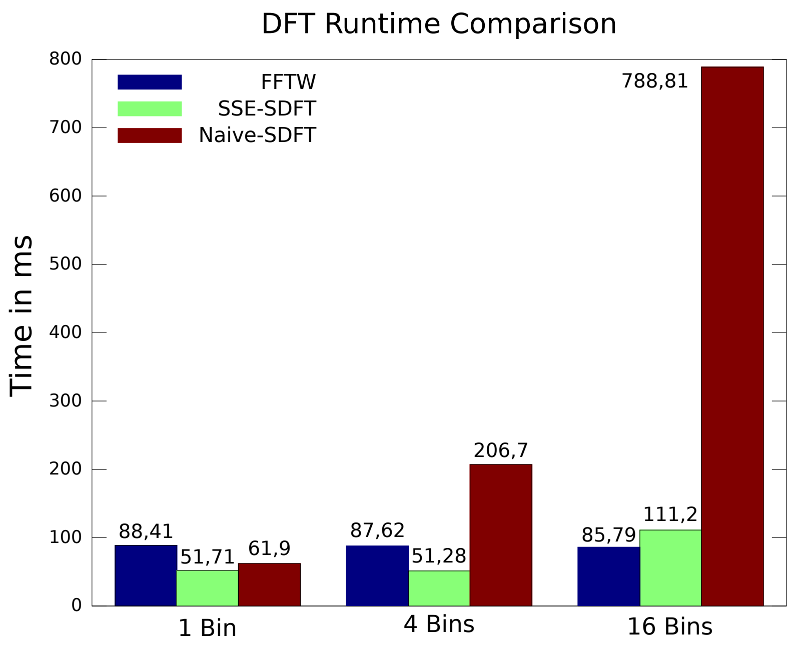


Figure 5.3: Runtime comparison for thre different methods. The bars are based on the mean of 100 runs for a 640×480 input frame and pixel-wise dft calculation.

On the right, the comparison for 16 bins is the most representative to show the advantages of the FFTW over the SSE method, as well as those of the SSE method over the naive implementation. The FFTW is computational superior in finishing this task, as it is designed to always calculate all bins of the DFT. Being slightly slower, the SSE implementation can be, in theory, faster than the FFTW for the case of continuous data. The reason for being not is simply the overall code performance. While the FFTW represents an internationally accepted, fast library optimized by many experts, the presented SSE code was conceptually designed in line with this thesis. Considering this disparity, the SSE method shows a large increase in performance compared to the naive algorithm. On the left the naive implementation is very close to the SSE approach, which performs best here. This is caused by the design of the SSE method, which always calculates at least four bins (see Section 5.11.3). In case of calculating only one bin like in the graph on the left, the current SSE algorithm wastes the time it could have used for more than twice the number of operations. For this reason the run with four bins shown in the middle is provided to give a better impression about the relative runtime behavior. To finish this section, Table 5.3 provides the parameter setup for which the system was able to work in real-time. Only the parameters, which significantly influence the systems performance are listed. For these values the system was able to run at ≈31.529, which is the average value of 226 runs after the initialization of the particle-filter.

Table 5.3: Parameters for Real-Time Performance

Parameter	Shortcut	Description	Value
`--iterations`	`-i`	RANSAC-Iterations	140
`--normalization-threshold`	`-w`	ϵ-value around target value	0.05
`--normalization-target`	`-o`	Target value for normalization. At this value we expect an LED.	1.3
`--particle-number`	`-n`	Number of particles in particle-filter	140
`--patch-size`	`-x`	Size of an “image patch” (c.f.ROI) around each LED for use in particle-filter	2×2
`--sample-size`	`-s`	The input image is scaled down to this size before it is processed	160×120

6 Experiments

In the following I want to document which kind of tests were performed, what setups were used and the drawn conclusions. In case of the stationary object, this is done in form of *boxplots* to provide a quick visual outline about the system’s performance. For a moving object, the algorithm did not perform well enough to provide a direct, metric comparison with the ground truth data, which was collected for this purpose. Instead I systematically examine all parts of the system to detect possible error sources. Finally a hypothesis is provided why the current implementation was not able to track in motion.

6.1 Ground-Truth

In order to check how well the algorithm performs I used a separate, commercial tracking system. It provides a pose tracking resolution in the range of millimeters though this is dependent on the quality of the manual system calibration. That resolution is reached by multiple cameras observing the scene at the same time. Both, this system and the system presented in this thesis recorded the same situation at the same frame rate. To synchronize the results I performed a rotation on the y-axis of the cart,[17] which could be identified in both logs.

The commercial system is a professional motion capturing system from *Qualisys*. It is based on autonomously acting cameras recording a scene, that is illuminated by a grid of infrared LEDs arranged around the lens of each camera. This light is subsequently reflected by passive marker spheres, which are mounted on the observed object. The reflectors are *retroreflective*, meaning

[17]Rotating around the y-axis means raising the hitch of the cart (see Figure 4.14)

most of the light arriving is reflected in the direction of the light-source. As the markers do not differ from each other, the system uses geometric constraints to resolve the correspondence. Therefore multiple cameras are required to observe the target from different perspectives. At the same time, the cameras process the marker position on their image plane, and the global tracking software joins the information of all devices to evaluate the final object pose. Due to the fact that there are three cameras in my setup, it does not matter if a tracked point is lost in one frame by occlusion.

Figure 6.1: Promotion picture of Qualisys Pro Reflex™ MCU 1000 motion capturing camera[18]

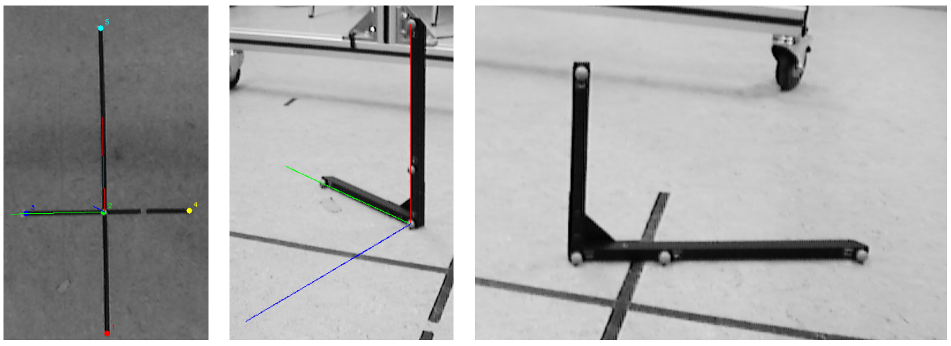
Parameters of the used setup

- 3x Pro Reflex™ MCU 1000 capturing devices (Figure 6.1)
- Uses the *World*[19] coordinate system (see *Qualisys L-Wand* on page 98)
- Captures 30 fps

Both tracking systems refer to the coordinate origin of the Qualisys system. For a good calibration of the camera a planar pattern on the floor, namely a cross is used. It is built by five points and can be seen in Figure B.2 (name: *Calibration Cross*). The Qualisys calibration pattern (*L-Wand* seen in Figure 6.2), a metal tri-square with fixed markers is placed on top of this cross with point *C1* (see Section B.2) located at the center of the calibration-cross. In the following, the standard calibration sequence for the Qualisys system was performed by moving the *wand*, a T-shaped metal with known geometry and two markers across the scene. After the system is calibrated, four markers are placed on the end points of the cross' bars and one on the intersection. This cross is visible in the range of my system's camera. Since the cross is high in contrast, its end points are easy to spot. An ideal calibration *World* $\mapsto$ *FWCam*[19] was calculated manually, by mapping the observed image points to the *World* coordinates of the tracking system. Using those coordinate pairs, the relative pose of the *FWCam* (FireWire Camera of the proposed system) to the motion capturing system's coordinates is propagated. The system reports the center of a sphere as location, so the point of origin hovers above the floor at about 15 mm in positive z-direction.

[18]Image source: `http://www.qualisys.com/archive/image_products_small/Proreflex%20big.jpg`

[19]See Table 8 for a list of coordinate systems



(a) Calibration Cross (b) Usable calibration with L-Wand (c) Wrong calibration with L-Wand

Figure 6.2: Calibrations with cross and L-Wand. Each of the images contains a 30×30×30 mm coordinate system, drawn in points of 1 cm distance representing the coordinate system it shall calibrate (red x-axis, green y-axis, blue z-axis). The additional colors in (a) represent backprojected IR-markers of the Qualisys system.

In Figure 6.2(a) the above mentioned cross is visible together with the back-projected positions of the markers. The calculated positions are good and yield a back-projection-error in the sub-millimeter range if converted from *World* $\mapsto$ *FWCam*[20] and back *FWCam* $\mapsto$ *World*[20]. Figures 6.2(b) and 6.2(c) show alternative calibrations using the L-wand. As mentioned in Section 4.6 this shape is not usable for calibration of the camera's relative pose to an object. The reason why it works for Figure 6.2(b) might be the spacial displacment of the points towards the camera. In contrast to the former figure, Figure 6.2(c) is recorded with the plane spanned by the wand almost parallel to the image plane of the camera. Without spacial information, the resulting values are completely wrong, thus the axes do not even appear in the image.

6.2 Initialization of a Stationary Object's Pose

6.2.1 Intial Guess

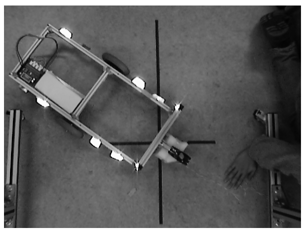
Figure 6.3(b) shows the quality of the pose-initialization which resulted from applying my pose-tracking system to the pictured setup scenario. The values in Figure 6.3(b) are based on a stationary object observed from the perspective shown in Figure 6.3(a). The three boxes show how well the system matched the particular axis coordinate for LED 1 (point *D1* in Figure B.1). Not shown in this figure are three frames with massive outliers. As orientation for the quality of the numbers, the magenta position marks the motion-tracking's mean value. Both, the boxplot and the mean value, are generated from 27 subsequent values taken after the initialization of the Fourier input buffer. The observed values are acceptable for the x and y axis with a distance of $|\overline{X}_{\text{thesis}} - \overline{X}_{\text{mocap}}| = 7.6940\,\text{mm}$ for the x-axis and $|\overline{Y}_{\text{thesis}} - \overline{Y}_{\text{mocap}}| = 9.4070\,\text{mm}$ for the y-axis, meaning an average error of 1 cm. Not shown in this figure is an amount of approximately 5% outliers with deviations above 1 m, for which the value was unusable. Their number can

[20]See Table 8 for a list of coordinate systems

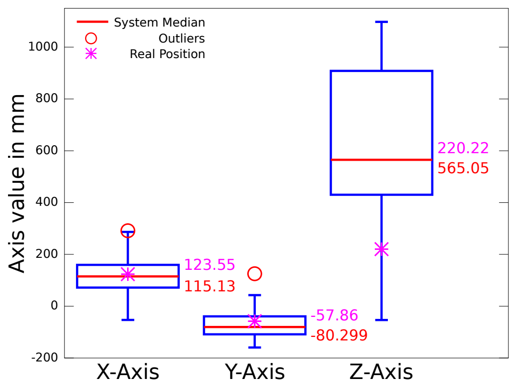
be decreased by increasing the RANSAC performance and using a higher number of iterations in this algorithm. Although the results still show a big error in terms of robotic navigation, it should be noted that these errors are a result of a chain of several factors.

- The LEDs were not arranged completely symmetric on the cart, since the blueprints provided in this thesis were created for this thesis, after the cart was built externally. There is no ground-truth available for the carts dimensions.
- All measurements for the LED's distances were made by hand with a sliding caliper or plain ruler. Even the LED-positions generated from the motion-capturing system are influenced by this error, as the interpolation between the markers' and LEDs' position uses this data.
- The calibration uses 4 points, which is the minimal required number of points to perform a 3D$\mapsto$2D homography based pose estimation. In his original paper [Zhang, 2000], the author of the underlying calibration method used patterns having between 100 and 300 features.
- Zhang also shows that the calibration error is very high for the first frame and decreases strongly after multiple different views of the target are available. This is not possible in the fixed, single-camera setup that was used.

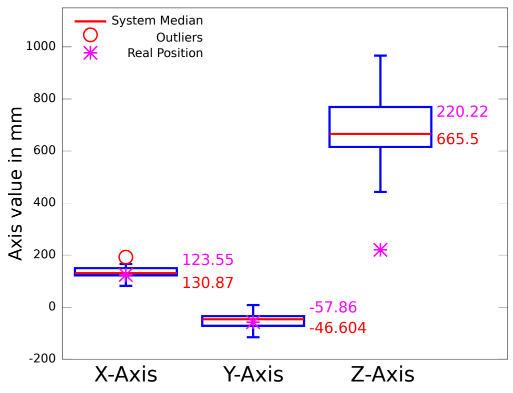
In contrast to the deviation of the x and y values, the large deviation on the z-axis can be easily seen. The reason for this is the artificial setup, which had to be chosen because of the fixed LED arrangement on the cart. The camera view is from directly above the cart. Therefore raising or lowering the device will cause only very small changes in the calibration pattern (cf. Figure 6.2(a)) by narrowing the points a bit. The amount of 140 iterations for the RANSAC algorithm described in Section 4.4.4 is small enough to provide the other parts of the system with enough time-resources to finish until the time slot for the current frame has expired. In comparison Figure 6.3(c) shows the results for an increased number of 800 iterations for the RANSAC algorithm. While it provides slightly better results, it also fails in correcting the z-axis error and goes beyond for realtime performance.



(a) Camera image showing the perspective



(b) Deviation of each axis values compared to the motion capturing system (140 Iterations)



(c) Deviation of each axis values compared to the reference system (800 Iterations)

Figure 6.3: Comparison of the system's result to the reference system

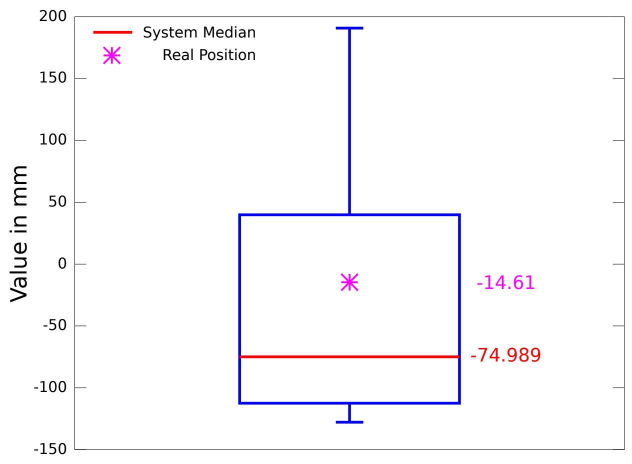
6.2.2 Particle-Filter

Based on the initialization results described above, the particle filter is initialized. Figure 6.4 shows that the values are generally more spread. It is caused by the filter, generally averaging weighted over all particles. This shows that there must be many particles striving into random direction. However, the spread depends on the used noise values. In the tests performed those were adjusted to give a reasonable estimate for the carts movement (see Table 6.1).

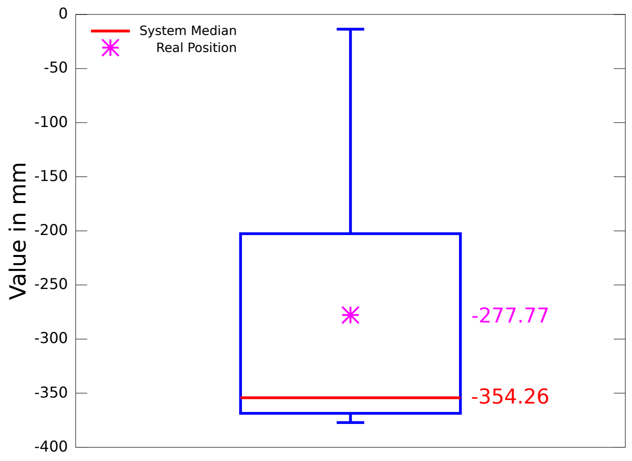
Table 6.1: The used noise values depict the average velocity deviation per second, which is applied in each movement step as product with $\sqrt{\Delta\, t}$, with $\Delta\, t$ as difference in ms between the current, and the last measurement.

Rotation in (°/s)			**Translation in (mm/s)**		
X	**Y**	**Z**	**X**	**Y**	**Z**
0.2	0.2	1	25	7	2

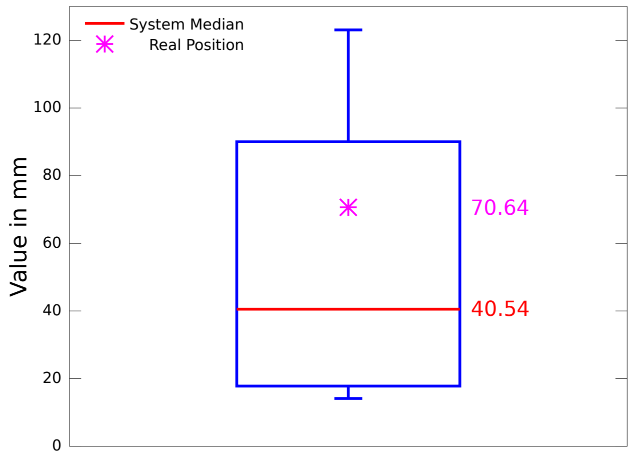
They describe the typical movement of the cart, which can move fast on the x-axis of the *Hitch* coordinate system, only moderately fast on its y-axis, and very little on the z-axis. Similary the rotation values were adjusted to model normal movement on a planar ground rotating about 1° on the z-axis. The two small values for the y- and z-axes allow the filter to compensate for an incorrect pose.



(a) X-Axis



(b) Y-Axis



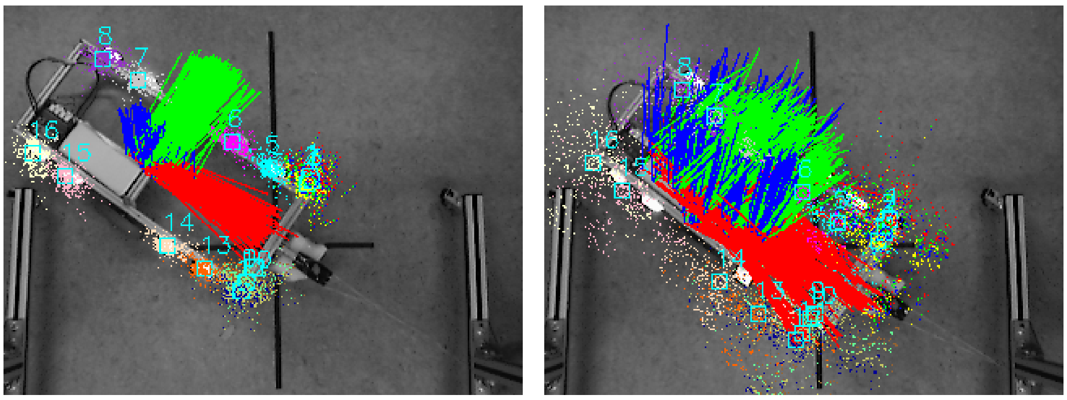
(c) Z-Axis

Figure 6.4: The figures show for each component of the cart's pose $\vec{P}_{cart} = (x_c, y_c, z_c)^T$ its distribution value distribution for 100 runs of the particle-filter. In (a), the system measured values for x_c in the range depicted by the boxplot, while the mean value of the commercial tracking system showed is shown by the star.

While the particle-filter spreads better around the target values it also spreads the values of the x- and y- axis, which were better in the initialization. This is a normal effect based on probabilistic estimation, which combines all particles to form a hypothesis. Although a single measurement might worsen, the overall hypothesis for the extrinsic parameter can improve. In case of the presented system it is likely to happen that the hypothesis getting worse for one particular LED enhances those of other ones.

6.3 Estimation of a Moving Object's Pose

In order to test the performance of the system in motion, multiple runs for lateral and horizontal motion relative to the camera's pose, as well as simple rotation were performed. Interpreting the values in general shows problems in the accuracy. The system was designed to track the cart in the image and provide a plausible position after movement. However, after converging on the initialized position and following the pose for about 20 cm, the filter stops providing a usable hypothesis. At least at the current state of implementation, it does not allow an accurate navigation of a robot. Figure 6.5 shows a plot of the cart performing straight motion. The particles are drawn as tuples of red x-axis, green y-axis and blue z-axis. The situation shows the point in time where the filter loses the cart and starts providing wrong hypotheses.



(a) Particles converging towards standing cart

(b) Particles losing the cart

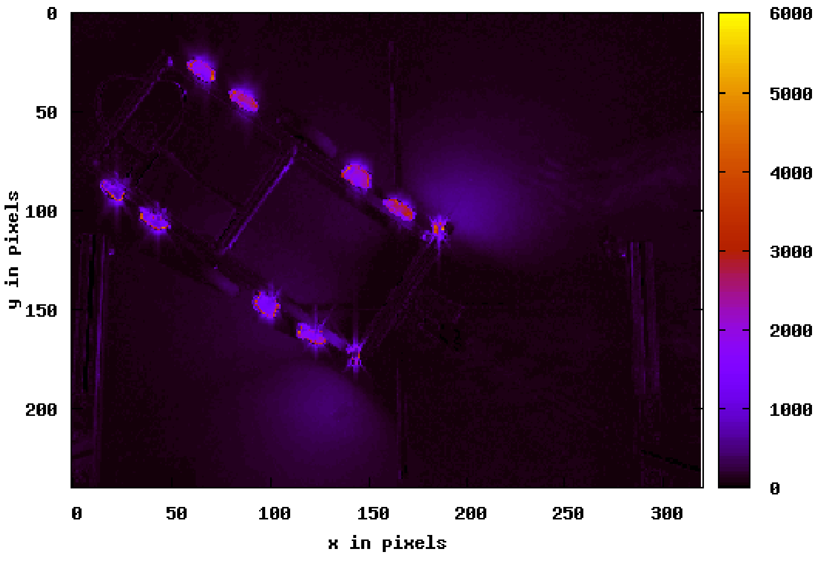
Figure 6.5: Particles spread around the hypothetical position of the cart. They are shown as red x-axis, green y-axis and blue z-axis each. Additionally the LEDs and the ROIs are shown. In Figure (a) the filter converges towards a usable pose while the cart is stationary. In contrast Figure (b) shows that the filter looses track of the cart after a short period of time in linear motion on the x-axis.

Summarizing, there were varying approaches to increase the performance of the algorithm. Each of them tried to improve a single aspect of the system, to approach the possible error from multiple points of view. For example the method to weight a particular particle was changed several times, because it is one of the most important regions for the cooperation of the segmentation and estimation parts of the software. It has to provide a measure for the particle filter whether the pose of this particle is good regarding the observed frames or not. To operate on the image, again, the back-projections[21] of the LEDs into the image are used. Possible variants:

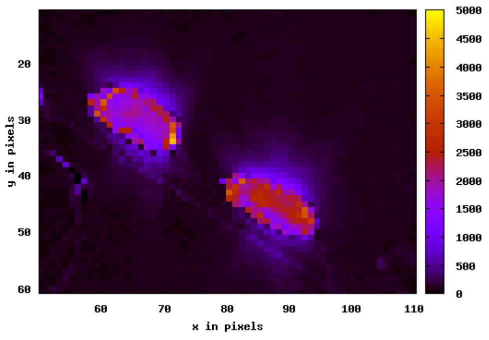
- Sum of normalized magnitudes for the maximum bin of each pixel's spectrum.
  Problems: The value of an unsought LED's bin might corrupt the value.
- Sum of pixels responding. Accept only the pixels where the bin index $i = argmax(|\hat{X}_i|)$, where i is also the index of the ROI defined for LED$_i$.
  Problems: The quality of a frequency response is not considered.
- Another hypothesis is that good regions are those with an LED in the center, not on the edge. To enforce this policy the distance of a pixel in the ROI to the center of that ROI is calculated. This is similar to the kernel used in [Yamazoe et al., 2004].

All of those measures were implemented and checked, but none of the ratings worked in a way, which made the overall approach applicable. In consequence, possible error sources were analyzed to find the reason for the failure of the approach. In the beginning unit-tests were used to verify critical sections of the software containing perspective mappings or other matrix operations. In the course of this, many software problems could be excluded. Subsequently the thresholding values were evaluated. Figure 6.6 visualizes the values observed by the system after the DFT is completely filled. Generally the visible values support the assumption that the LEDs should be weighted higher than other regions in the image. While the plain use of the frequency-domain magnitude in 6(a) has the advantage of being more distinct regarding the LEDs, it is not possible to apply the same threshold on different image sets. This means, if a particular magnitude strength is expected for an LED, the LED will possibly be ignored as the background changes. The reason is that the detected frequency on it's own will be the same, but the magnitude decreases as the contrast between background and lit LED decreases. In contrast, the values for LEDs shown in 6(d) are not that easy to distinguish from background reflections, but are invariant regarding the intensity as stated in Section 4.3.4. The normalized values, which are preferred due to their illumination invariance, thus need a range threshold instead of a strict barrier. This means a target value $a \approx 1.3$ is defined for the normalization and another value $b \approx 0.1$ defines the allowed deviation from that target value.

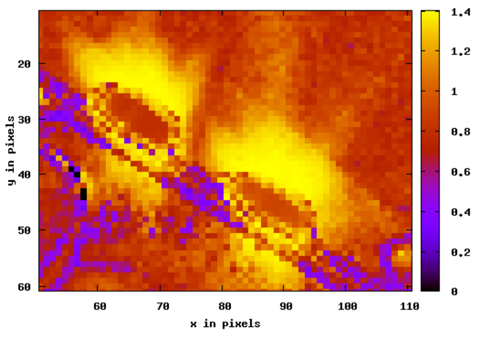
[21] See Figure 4.16 for an explanation of LEDs-back-projections to the image.



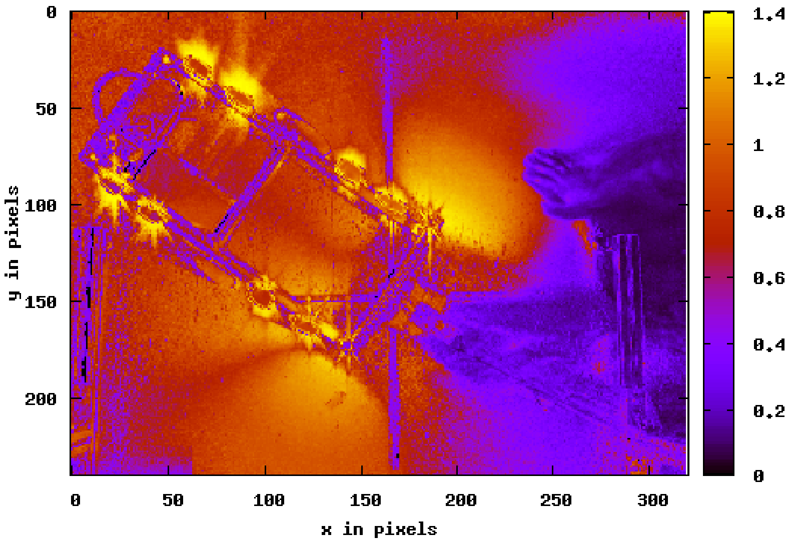
(a) Segmentation based on Frequency Magnitude



(b) Zoomed upper left LEDs of (a)



(c) Upper left LEDs of (d)



(d) Segmentation based on Normalization

Figure 6.6: The above images give an intuition about the values used for pixel-segmentation. The values in (a) represent the frequency magnitude, whereas those in (d) are based on the normalization introduced in Equation (4.4).

After testing the segmentation, I tried to visualize the values computed internally by the particle filter. Therefore I took a set of input frames and initialized the system to create ROIs at about the position of the LEDs. The values observed in Figure 6.7 revealed the expected behavior. The good ROI is weighted better than the partly correct one, and both are much better than random data.

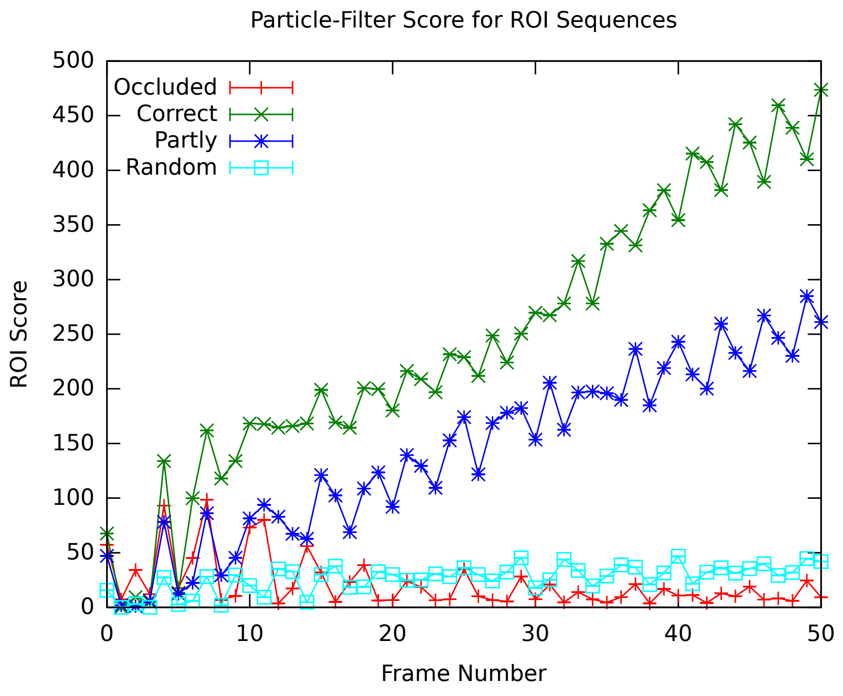


Figure 6.7: Scores of a ROI on the correct position inside the image, matching almost the LED's position, and on random data. The occluded sequence means the projection of an LED which is not visible and thus will provide bad data. The score is the value used to weight the particles.

In the following I observed the behavior of the score for a good sequence, in which wrong data is inserted. The simplest case for wrong data is a dark frame, or possibly image background, where a bright LED was expected. This situation is shown in Figure 6.8, where at frame 100 a bright pixel was expected, but a dark one provided. This situation might occur if a particle moves away from a fitting LED position. In general, this is just an example of what will happen inside the filter, simplifying the way the score is calculated. However, what can be observed is that the score decreases instantly at frame 100. This behavior is desired, since a particle not fitting to the expected data should be devalued. What is not undesired, however, is the fact that that value remains lower for the next 70 frames, where in this sample $N = 70$ is the resolution of the DFT.

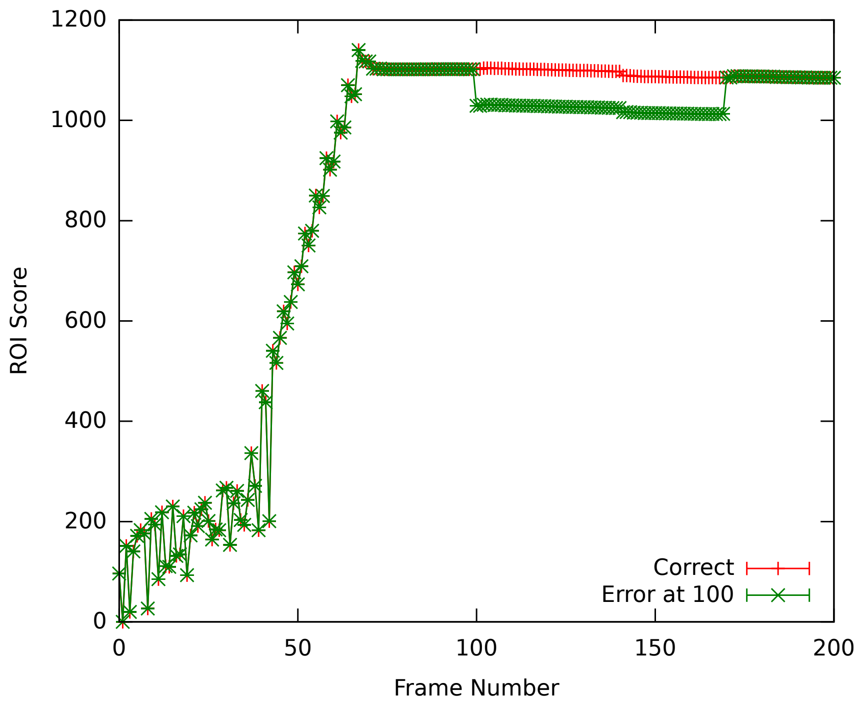


Figure 6.8: Score for a ROI analogous to Figure 6.7. The red line shows how the score changes if one wrong frame is placed at frame 100.

An explanation for this behavior is quickly identified by reviewing the definition of the frequency-domain magnitude in Section 3.1.2. The intensity values ι of the wrong frame are pushed on the input buffer of the particle at frame 100. The buffer represents the frequency-domain signal's property of being composed of a sum over the input signals. The new value in the buffer instantly changes the value of the DFT. Furthermore, as the frequency-domain magnitude is defined internally as a sum over these values, the effect of that value stays constant. Moreover the constant offset moves from the end of the circular buffer to the beginning in 70 frames after which it has no more influence on the signal anymore. That explains why after 70 frames, the score returns back to a higher value.

Looking at the feedback approach (see Figure 4.18) to estimate a pose, this can lead to undesired effects. For example a bad frame devalues instantly, but a good frame does not increase the weight in the same way. Actually every frame after the erroneous one is exactly the same for both sequences drawn in Figure 6.8. Despite that, the particle is not increased in weight because the bad frame is still inside the circular buffer. Quite the contrary, if a second bad frame follows the value will get even worse, although it has provided good values for almost the same time as another particle.

All this conflicts with how the particle-filter compares the weights of the particles. The particle-filter is fair with respect to one frame. This means, after resampling, which is done after every frame, each particle has the same chance of getting into the new sample (see Figure 3.4). In contrast, the scoring via the DFT is fair with respect to a complete cycle of input data. The difference between the time frame considered in the particle filter and that in the DFT, can cause the score of good particles to degenerate until the DFT has raised it, after the bad frame has left the buffer.

Nonetheless, the presented hypothesis for the failure of the approach does not assume the above mentioned differences of considered time frames as the only reason. It rather shows a conceptual problem of using frequency analysis for rating a dynamic system. As previously stated by [Allen et al., 2001], tracking systems should use sampling rates of above 40 Hz. The found problems support this statement as a higher sampling rate means faster information updates, thus smaller effects of the above described degeneration situation. Another consequence might be to orientate the approach on the feature vectors presented by [Koch et al., 2008]. This allows smaller DFT resolutions, which might also reduce the effect of the problem described.

6.4 Results

Summarizing, the performed tests show as a "proof of concept" that it is possible to recognize an object with frequency information. For stationary objects the tests show good results, having a maximum of $\approx$1-6 cm deviation from the position determined by a commercial motion tracking system. Albeit this is only true if the fact is ignored that only bird's eye view recordings were possible due to the mentioned problems with the L-shaped LED arrangement on the side of the cart. For stationary objects, the approach works well and can provides a precise estimation of an object's attitude relative to the camera.

The goal of robust frequency recognition was completely reached using the implemented Fourier approach. The system can perform the necessary calculations in real-time but is limited to a restricted pixel range in this case. A resolution of 160×120 pixels lets the system perform in real-time. This resolution can be used to spot the target object in the image and later-on split to inspect special ROIs in more detail.

From the segmented pixels a position can be calculated using the RANSAC approach with rated back-projection of the LEDs. The second goal of this work was therefore reached, the generation of a pose out of registered blinking-patterns.

Finally, tests in motion were performed. The laborious setup for reaching ground-truth has shown weaknesses of the current system in registering motion. While in the stationary case, the particles converge in tolerable limits, the correspondence of ground-truth and the data from the system showed a significant deviation when the object being tracked is in motion. A hypothesis for the reasons of those problems is given in Section 6.3. Generally the result of this work shows the not the maximum possible accuracy of the supposed method, because there were only a minimal amount of features available to identify the searched object, namely four LEDs in a plane.

7 Conclusion

The presented method is effective for solving the correspondence problem using a single standard camera and inexpensive high energy LEDs. The final solution additionally requires no lighting dependent configuration of thresholds assuming a minimum image quality, which can be provided by automatic exposure-time and aperture adjustments of current camera systems. Apart from the capturing device only some electronics to drive the LEDs is needed. A piece of software was developed, to calculate a multiplicity of frequency analyses on consumer hardware in real-time. The described approach yields the ability for large scale Fourier transforms in case of continuous data. This allows an unusual interpretation of pixelwise frequency data. The tests of the implemented probabilistic approach reveals problems in the robustness of estimating a moving object's pose. However the system is, according to the initial motivation, much less sensitive to lighting influences. Nevertheless, this robustness comes at the price of less robustness in motion.

7.1 Lessons Learned

As the focus of this work was on the development of the method itself and the software required to implement it, the used hardware was not modified to a large extent. While this allowed broad research and development of multiple approaches to solve the faced problem, some fundamental issues had an impact on the quality of the product, which were not solvable by smarter software. In general the decision of using only the minimal amount of features is questionable. Although the placement of the LEDs was decided during the thesis, the development status at that time was insufficient to make a reasonable decision. It would have been better to completely ignore the arrangement until the final tool-chain was developed.

Furthermore the complete development of the HMM-approach before its functionality was proved, was disadvantageous. As it was a first idea to solve the problem, much time was consumed with the result of a dead-end related to the fundamental Viterbi algorithm. Resulting from this problem the alternative Fourier approach was deeply investigated before using it. Only after the the needed theory was completely overseen and a supportive method to reduce its runtime (the S-DFT) was found, I started implementing it. As a result however the time consumed by the development of the HMM approach was missing in the late phase of improving the final work. Similar to that, the problem regarding the overlooked invalid arrangement of three features on a line when calculating a homography points in the same direction. Comprehensive checking of the necessary concepts in the way a deep-first search saves time and contributes to the overall quality of the implemented approach. In addition this yields the advantage of a complete catalog of requirements allowing straighter development.

Retrospectively seen, the decision to arrange the whole software as modular library was very good. Uncoupling specific functionality such as the multiple transformation of coordinates for the comparison of ground-truth data acquired via shifted markers was no struggle. Moreover I profited from the library design when the system scaled. For example the `Sampler` was designed as a wrapper to allow ROIs in the `DFTSampler`, which has no knowledge of this.

Additionally the work pointed out the benefits of the smart-pointer concept from the boost-library, which was very supportive. Memory management is one of the most tedious tasks to deal with using a language with direct access to the memory. Using the smart objects makes it possible, to a certain extent, to eliminate software errors due to memory mismanagement. This saves a lot of time for research, as the effects of memory corruption are difficult to spot.

7.2 Future Prospects

The most essential future task would be the stabilization of estimation in motion. Considering the different problems which were encountered, one possible starting point can be the use of direct intensity information instead of frequency-measures during the motion estimation. By the initialization using the proposed frequency-based approach a phase-offset between the blinking patterns and the capturing rate of the camera can be estimated, which allows to forecast the expected intensity value in a certain image region of an assumed LED. The increased quality of the measurement step in the particle-filter might increase the usability of the motion estimation.

In a second step the performance of the system can be increased. Both, the RANSAC and the particle-filter methods provide a variable quality/performance trade-off. In other words the accuracy of both methods can be increased using more computational resources. One possibility for further performance increases is given by extended parallelization. As the SIMD instructions already increased the performance strongly, more vectorization is possible on other hardware. Using the Graphics Processing Unit (GPU), arrays of many processors which can operate in parallel, are available on consumer hardware. The scientific field of these applications is called General Purpose GPU-Programming (GP-GPU) computing. In order to support a common syntax Khronos Group, a committee funded by the leading graphics hardware companies, have recently released *OpenCL* (Open Computing Language). This is a programming language, whose syntax can later-on be translated into instructions understood by manufacturer-specific drivers. Alternatively there are many other options of hardware supporting extended vectorization, such as the mobile devices presented in [Koch et al., 2008].

Further performance increments can be reached by reducing the computational effort for the homography estimation. A rough estimation of a possible fitting homography might be enough to let the RANSAC algorithm converge towards a usable initialization value. In addition there are many aspects supporting the system in a smaller scale, which in sum might have a noticeable influence. Approaches in this direction include a clean construction of a new target object, distortion of ROIs according to the assumed pose of the corresponding particle or supportive mechanisms like optimization algorithms, which can optimize the configuration of a particle filter.

References

[Allen et al., 2001] Allen, B. D.; Bishop, G.; and Welch, G. (2001). ”Tracking: Beyond 15 Minutes of Thought”. In *SIGGRAPH*. University of North Carolina at Chapel Hill Department of Computer Science. 9, 37, 41, 80

[Baker, 1987] Baker, J. E. (1987). ”Reducing bias and inefficiency in the selection algorithm”. pages 14–21. 20

[Barca et al., 2008] Barca, J. C.; Rumantir, G.; and Li, R. (2008). *Computational Intelligence in Multimedia Processing: Recent Advances*, volume 96/2008, chapter Noise Filtering of New Motion Capture Markers Using Modified K-Means, pages 167–189. Springer. 8

[Baum et al., 1970] Baum, L. E.; Petrie, T.; Soules, G.; and Weiss, N. (1970). ”A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains”. *The Annals of Mathematical Statistics*, 41(1), pp. 164–171. 11

[Birbach et al., 2008] Birbach, O.; Kurlbaum, J.; Laue, T.; and Frese, U. (2008). ”Tracking of Ball Trajectories with a Free Moving Camera-Inertial Sensor”. In *Proceedings of the RoboCup International Symposium, Suzhou. RoboCup International Symposium, located at The 12th RoboCup International Competitions and Conferences, RoboCup-2008 Suzhou, July 15-18, Suzhou, China*. o.A. 47

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV*. O’Reilly Media Inc. 7, 15

[Brodowski and Dongili, 2005] Brodowski, D. and Dongili, M. (2005). *Manpage of the cpufreq-info tool*, 0.1 edition. Contact: linux@brodo.de, malattia@gmail.com. 95

[Brown, 1971] Brown, D. C. (1971). ”Close-range camera calibration”. *PHOTOGRAMMETRIC ENGINEERING*, 37(8), pp. 855–866. 60

[Burgard et al., 2009] Burgard, W.; Stachniss, C.; Grisetti, G.; Bennewitz, M.; and Plagemann, C. (2009). ”Material for “Introduction to Mobile Robotics” - Lesson held at Albert-Ludwigs-Universität Freiburg”. 21

[Cooley and Tukey, 1965] Cooley, J. W. and Tukey, J. W. (1965). ”An Algorithm for the Machine Calculation of Complex Fourier Series”. *Mathematics of Computation*, 19(90), pp. 297–301. 14, 89

[Dawes et al., 2009] Dawes, B.; Rozental, G.; and Maddock, J. (2009). *Boost 1.41.0 Library Documentation*. 54

[DiBona et al., 1999] DiBona, C.; Ockham, S.; Mark Stone, Behlendorf, B.; Bradner, S.; Hamerly, J.; Mckusick, K.; O’Reilly, T.; Paquin, T.; Perens, B.; Raymond, E.; Stallman, R.; Tiemann, M.; Torvalds, L.; Vixie, P.; Wall, L.; and Young, B. (1999). *Open Sources: Voices from the Open Source Revolution (O’Reilly Open Source)*. O’Reilly. 89

[Douxchamps et al., 2009] Douxchamps, D.; Peters, G.; and Others (2009). ”libdc1394 Homepage”. 52, 54, 90

[Eaton, 2002] Eaton, J. W. (2002). *GNU Octave Manual*. Network Theory Limited. 14

[Feldman et al., 2002] Feldman, J.; Abou-Faycal, I.; and Frigo, M. (2002). ”A fast maximum-likelihood decoder for convolutional codes”. *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, 1, pp. 371–375 vol.1. 32

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). ”Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. *Commun. ACM*, 24(6), pp. 381–395. 17

[Forney, 1973] Forney, G. D. (1973). ”The viterbi algorithm”. *Proceedings of the IEEE*, 61(3), pp. 268–278. 11

[Free Software Foundation, 2008] Free Software Foundation, I. (2008). *A GNU Manual - GCC 4.4.3 Manual*. 54

[Frigo, 2003] Frigo, M. (2003). *FFTW 3.2.2 User Manual, What FFTW Really Computes - The 1d Discrete Fourier Transform (DFT)*. Massachusetts Institute of Technology, 3.2.2 edition. 14

[Frigo and Johnson, 2005] Frigo, M. and Johnson, S. G. (2005). ”The Design and Implementation of FFTW3”. *Proceedings of the IEEE*, 93(2), pp. 216–231. Special issue on “Program Generation, Optimization, and Platform Adaptation”. 14, 66, 89

[Furniss, 2000] Furniss, M. (2000). ”Motion Capture: An Overview”. *Animation Journal*, pages 68–82. `http://web.mit.edu/comm-forum/papers/furniss.html`. 48

[Galassi et al., 2009] Galassi, M.; Gough, B.; and Davies, J. (2009). *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd. 54, 89

[Gonzalez and Woods, 2006] Gonzalez, R. C. and Woods, R. E. (2006). *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 5

[Intel, 2007] Intel (2007). *Intel C++ Intrinsic Reference*. Intel Corporation. 63

[Intel, 2009a] Intel (2009a). *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Intel Corporation. 63

[Intel, 2009b] Intel (2009b). *Intel 64 and IA-32 Architectures Software Developers Manual, Volume 1: Basic Architecture*. Intel Corporation. 63

[Jacobsen and Lyons, 2003] Jacobsen, E. and Lyons, R. (2003). ”The sliding DFT”. *Signal Processing Magazine, IEEE*, 20(2), pp. 74–80. 15

[Jacobsen and Lyons, 2004] Jacobsen, E. and Lyons, R. (2004). ”An update to the sliding DFT”. *Signal Processing Magazine, IEEE*, 21(1), pp. 110–111. 15

[Jansen et al., 2007] Jansen, C.; Steinicke, F.; Hinrichs, K. H.; Vahrenhold, J.; and Schwald, B. (2007). "Performance Improvement for Optical Tracking by Adapting Marker Arrangements". In Zachmann, G., editor, *VR Workshop on Trends and Issues in Tracking for Virtual Environments*, pages 28–33. IEEE, Shaker-Verlag. 9

[Kalman, 1960] Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems". 19

[Keenan and Hosack, 1989] Keenan, W. and Hosack, H. (1989). "A channel-stop-defined barrier and drain antiblooming structure for virtual phase CCD image sensors". *Electron Devices, IEEE Transactions on*, 36(9), pp. 1634–1638. 25

[Koch et al., 2008] Koch, M.; Zivkovic, Z.; Kleihorst, R.; and Corporaal, H. (2008). "Distributed Smart Camera Calibration Using Blinking LED". In *ACIVS '08: Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 242–253, Berlin, Heidelberg. Springer-Verlag. 9, 35, 80, 82

[Martin and Hoffman, 2008] Martin, K. and Hoffman, B. (2008). *Mastering CMake 4th Edition*. Kitware, Inc., USA. 54

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). "The Monte Carlo Method". *Journal of the American Statistical Association*, 44(247), pp. 335–341. 19

[Oppenheim and Schafer, 1989] Oppenheim, A. V. and Schafer, R. W. (1989). "Discrete-time signal processing". 15

[Peleg and Weiser, 1996] Peleg, A. and Weiser, U. (1996). "MMX Technology Extension to the Intel Architecture". *IEEE Micro*, 16(4), pp. 42–50. 63

[Perry, 1990] Perry, T. S. (1990). "Biomechanically engineered athletes". *IEEE Spectr.*, 27(4), pp. 43–44. 9

[Rabiner, 1989] Rabiner, L. R. (1989). "A tutorial on hidden markov models and selected applications in speech recognition". In *Proceedings of the IEEE*, volume 77, pages 257–286. 10, 11, 26

[Raman et al., 2000] Raman, S.; Pentkovski, V.; and Keshava, J. (2000). "Implementing streaming SIMD extensions on the Pentium III processor". *Micro, IEEE*, 20(4), pp. 47–57. 63

[rn-wissen.de, 2008] rn-wissen.de (2008). "Hallo Welt für AVR (LED blinken)". 61

[Röfer, 2008] Röfer, T. (2008). "Region-Based Segmentation with Ambiguous Color Classes and 2-D Motion Compensation". (5001), pp. 369–376. 5

[Russell and Norvig, 2003] Russell, S. J. and Norvig (2003). *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall. 7, 19, 31

[Schepers, 2009] Schepers, H. M. (2009). *Ambulatory assessment of human body kinematics and kinetics*. PhD thesis, University of Twente, Enschede. 9

[Seward et al., 2008] Seward, J.; Nethercote, N.; Weidendorfer, J.; and Team, V. D. (2008). *Valgrind 3.3 Advanced Debugging and Profiling for GNU/Linux applications*. Network Theory Limited, 1 edition. 62

[Shannon, 1949] Shannon, C. (1949). ”Communication in the Presence of Noise”. *Proceedings of the IRE*, 37(1), pp. 10–21. 36

[Smith and Gelfand, 1992] Smith, A. F. M. and Gelfand, A. E. (1992). ”Bayesian Statistics without Tears: A Sampling-Resampling Perspective”. *The American Statistician*, 46(2), pp. 84–88. 19

[Smith et al., 2004] Smith, K. B.; Bik, A. J.; and Tian, X. (2004). ”Support for the Intel Pentium 4 Processor with Hyper-Threading Technology in Intel 8.0 Compilers”. *Intel Technology Journal*, 8, pp. 19–31. 64

[Smith, 1997] Smith, S. W. (1997). *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA. 7, 11, 12, 13, 14, 33, 36

[Sony, 2001] Sony (2001). *Technical Manual for Sony DFW-V500 and DFW-VL500*. Sony Corporation, 1.0 english edition. 52

[Sourceforge, 2008] Sourceforge (2008). ”OpenCV computer vision library”. `http://sourceforge.net/projects/opencvlibrary/`. Originally developed by Intel. 54

[Thrun et al., 2005] Thrun, S.; Burgard, W.; and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press. 8, 18, 19, 20

[van Heesch, 2009] van Heesch, D. (2009). *Doxygen Manual*, 1.6.1 edition. Contact: dimitri@stack.nl. 52, 54

[Van Overschelde and Wautelet, 2005] Van Overschelde, O. and Wautelet, M. (September 2005). ”Self-diffraction in a CCD camera”. *European Journal of Physics*, 26, pp. L15–L17(1). 25

[Westermann and Hauser, 1996] Westermann, B. and Hauser, R. (1996). ”Non-invasive 3-D patient registration for image-guided skull base surgery”. *Computers & Graphics*, 20(6), pp. 793 – 799. Medical Visualization. 9

[Xsens, 2009] Xsens (2009). ”The fascination for motion. Introduction about the beginning of motion capture technology”. `http://www.xsens.com/en/company/research/human_mocap.php`. Xsens Technologies B.V, Enschede, The Netherlands. 47

[Xu et al., 2008] Xu, D.; Li, Y. F.; and Tan, M. (2008). ”A general recursive linear method and unique solution pattern design for the perspective-n-point problem”. *Image Vision Comput.*, 26(6), pp. 740–750. 7

[Yamazoe et al., 2004] Yamazoe, H.; Utsumi1, A.; Hosaka1, K.; and Yachida, M. (2004). ”Geometrical and Temporal Calibration of Multiple Cameras by Using LED Markers for Image Synthesis”. In *ICAT*. 9, 76

[Zeng et al., 2008] Zeng, H.; Deng, X.; and Hu, Z. (2008). ”A new normalized method on line-based homography estimation”. *Pattern Recognition Letters*, 29(9), pp. 1236 – 1244. 49

[Zhang, 2000] Zhang, Z. (2000). ”A flexible new technique for camera calibration”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22, pp. 1330–1334. 60, 71

8 Glossary

Glossary

Notation	Description	Page List
aliasing	Aliasing is the effect which occurs if a wave form is sampled with a too low rate. Higher frequencies, which also belong to the original signal will appear in form of phantom signal fractions of lower frequencies in the frequency-domain	35
bin	A bin means one element in a histogram like figure. In the context of Fourier Analysis this means the value for one specific fraction of the sampling frequency, which is calculable with the current DFT's resolution	13, 14, 31–34, 37, 62, 64
BLOB	BINARY LARGE OBJECTS: In this context a group of connected pixels with a special property in the image.	6, 7, 38
grayscale	A grayscale image meant here is a monochrome image in shades of gray. Used here are 8 bit variants with values from 0 (black) to 255 (white), as well as a 16 bit version for differences between 8 bit images. Their values from $[-255 : 0]$ represent an 8 bit decrease in intensity, whereas the values from $[0 : 255]$ mean an increment	23–25, 28, 53
LED-pattern	Grid-Arrangement of LEDs	48, 61
monochrome	A monochrome image is an image in shades of only one color. Meant here are shades of gray ranging from black to white	20
particle	One local state hypothesis of the particle-filter algorithm described in Section 3.4	18
ROI	REGION OF INTEREST: A rectangular region in an image	43–45, 56, 58, 66, 68, 75, 76, 78, 80, 82

Acronyms

Notation	Description	Page List
C++	C++ programming language	52, 54, 63
C	C programming language	52, 63
ABG	ANTI BLOOMING GATES: Circuitry to stop the blooming effect in CCDs	24, 25
API	Application Programming Interface	52, 54
CCD	CHARGED COUPLED DEVICE: Light sensitive chip in a digital camera	8, 24
CPU	Central Processing Unit	62, 64
CT	Computer Tomograph	8
DC	DIRECT CURRENT: Average value of a signal in digital signal processing	32, 33
DFT	Discrete Fourier Transform	10, 11, 13, 20, 31–34, 36, 37, 43, 45, 46, 56, 57, 62, 66, 67, 76, 78–80
DSP	Digital Signal Processing	10, 13, 14, 31, 33
FFT	Fast Fourier Transform [Cooley and Tukey, 1965]	13, 14, 38
FFTW	Fastest Fourier Transform in the West [Frigo and Johnson, 2005]	13, 66, 67
FireWire	Apple trade name for the IEEE 1394 interface, used synonymously for the technology	20, 52, 54, 69
GNU	GNU IS NOT UNIX: A “backronym” (i.e. recursive acronym) [DiBona et al., 1999]	54
GP-GPU	General Purpose GPU-Programming	82
GPU	Graphics Processing Unit	82
GSL	GNU Scientific Library [Galassi et al., 2009]	54
HMM	Hidden Markov Model	6, 9, 10, 14, 20, 24, 25, 27, 30, 31, 56, 61, 81
HSI	HUE-SATURATION-INTENSITY: A color space defining colors by values a human would use to describe colors	5
ILED	Infrared-LED	47

Notation	**Description**	**Page List**
IRQ	Interrupt Request	61
LED	Light Emitting Diode	1, 5–9, 12, 20, 21, 23–25, 27–34, 36–38, 40, 43–49, 56–61, 66, 68, 70, 71, 75, 76, 78, 80–82, 87, 89
libdc1394v2	FireWire library [Douxchamps et al., 2009]	52, 54
MMX	MULTI MEDIA EXTENSION: SIMD instruction set introduced by Intel	63
MRT	Magnetic Resonance Tomograph	8
OpenCV	Intel Computer Vision library	15, 52, 60
PE	Processing Element	35
QVGA	QUARTER VIDEO GRAPHICS ARRAY: A shortcut for a resolution of 320×240 pixels used to describe display sizes	34, 37
RANSAC	Random Sample and Consensus	16, 18, 38, 40, 60, 71, 80, 82
RGB	RED-GREEN-BLUE: A color space defining colors by mixing the primary colors	5, 23, 87
S-DFT	Sliding Discrete Fourier Transform	6, 14, 32, 37, 63, 66, 81
SIMD	Single Instruction Multiple Data	38, 62, 63, 82
SMC	Sequential Monte Carlo	18
SSE	Streaming SIMD Extensions	63, 64, 66, 67
SUS	Stochastic Universal Sampling	19

Coordinate Systems

Notation	**Description**	**Page List**
Axis	The moved local coordinate system *Hitch* after the cart was relocated	42
FWCam	Coordinate system of the FireWire camera. Its origin is in the top left corner of the image, the x-axis points right, the y-axis down and the z-axis into space in the viewing direction of the camera. The system is right-handed	38, 42, 69, 70
Hitch	Local, right-handed coordinate system on the cart. The origin is located in the middle of the aluminum bar over the cart (see "Hitch Origin" in front view of Figure B.1). The x-axis points to the front through the hitch, the y-axis goes to the right and the z-axis towards the ceiling	38, 41, 42, 72
World	Global reference coordinate system. It is defined by the coordinate system of the motion capturing system. The "L-Wand" calibration tool defines the origin in point *C1* of Figure B.2. The x-axis points along the longer bar of the wand towards *C2*, the y-axis is defined along the smaller bar through *C3*, while the z-axis points upwards, towards the ceiling. This calibration tool is placed on a *Calibration Cross* of two tape-stripes on the floor	69, 70

9 Lists

List of Algorithms

List of Figures

[22]See Table 8 for a list of coordinate systems

A Appendix: Software

A.1 LED-Color legend

For debugging purposes a method was implemented to generate pictures from segmented images. In those images the color of a pixel depicts which LED is assumed to be represented by it. Therefore particular colors were assigned to the LED-indices as shown in the blueprint of Figure B.1. The following table assigns an RGB color value to each LED. Note that the relative representation of a frequency image scales down those values linearly depending on the signal strength of the pixel. As a result a pixel belonging to LED 1 (Red: 255, Green: 0, Blue: 0) with 50% of the maximum signal strength observed in one frame will get the relative value of (Red: $0.5 \cdot 255 = 127$, Green: $0.5 \cdot 0$, Blue: $0.5 \cdot 0$).

Table A.1: Colors showing a pixel's affiliation to an LED in the segmented image

LED-Index	Color-Name	Color	Red	Green	Blue
0	black	⬛	0	0	0
1	red	🟥	255	0	0
2	green	🟩	0	255	0
3	blue	🟦	0	0	255
4	yellow	🟨	255	255	0
5	cyan	🟦	0	255	255
6	magenta	🟪	255	0	255
7	light gray	⬜	200	200	200
8	purple	🟪	150	50	200
9	brown	🟫	150	100	50
10	mint	🟩	100	255	50
11	beige	🟨	200	200	100
12	dark blue	🟦	0	0	150
13	orange	🟧	255	100	0
14	peach	🟧	255	220	185
15	pig	🟥	255	180	200
16	light yellow	🟨	255	255	204

A.2 Experiments

The directory tree below shows the structure of all archives in the `analyse` folder. These archives contain the commands and input necessary to reproduce a certain output, which is also contained pre-built. This way the described experiments can be reconstructed for full transparency.

`/`

- `com/` ..... Contains meta-information about this test run.
  - `commands` ..... This file contains the command used to gain the output which is contained in the archive. All Parameters are in the long format to allow a natural language overview over the steps to reproduce the result.
  - `info` ..... Herein a timestamp, the revision number used as well as the output of the cpufreq-info [Brodowski and Dongili, 2005] tool give a clue about the environment settings which were used while this archive was filed.
- `dft` ..... In here, the visual representations of the DFT-Segmentation are filed. A black pixel on such an image means, that this pixel has no LED affiliation. Other colors match the legend described in Section A.1.
  - `region_<i>_<img>.tiff` ..... Those files contain the segmented output of the region with index <i> corresponding to the image file <img>.
  - `sampler_<img>.tiff` ..... Where the above mentioned region images contain only the segmented output of a certain region, this image is composed from the whole sample, containing all regions segmentation information placed at the right positions. Every image is related to the original frame named <img>.
- `pics` ..... This folder contains the original images used to calculate the output found in the other folders.
- `samples` ..... The images found here are the downscaled equivalents of the images found in pics.

Archives The following listing contains a short description of all archives with analyzed data, which are contained in the `analyse` folder of the thesis. See Section A.2 for a description of the structure inside those files.

`dft_1hz_false_positives.tar.bz2` By this test the effect of very low frequencies like 1 Hz are shown. After 4 frames there are a lot of false positives due to the fact, that the DFT-buffers contain only zero data and now are fed with the input of this very bright scene.

B Appendix: Technical drawings

The drawings in this section help understanding the results of the described algorithms. Except the technical drawing in B.3, they were created in line with this thesis. Below is a short description of the drawings including the sub-drawings on pages 97-98.

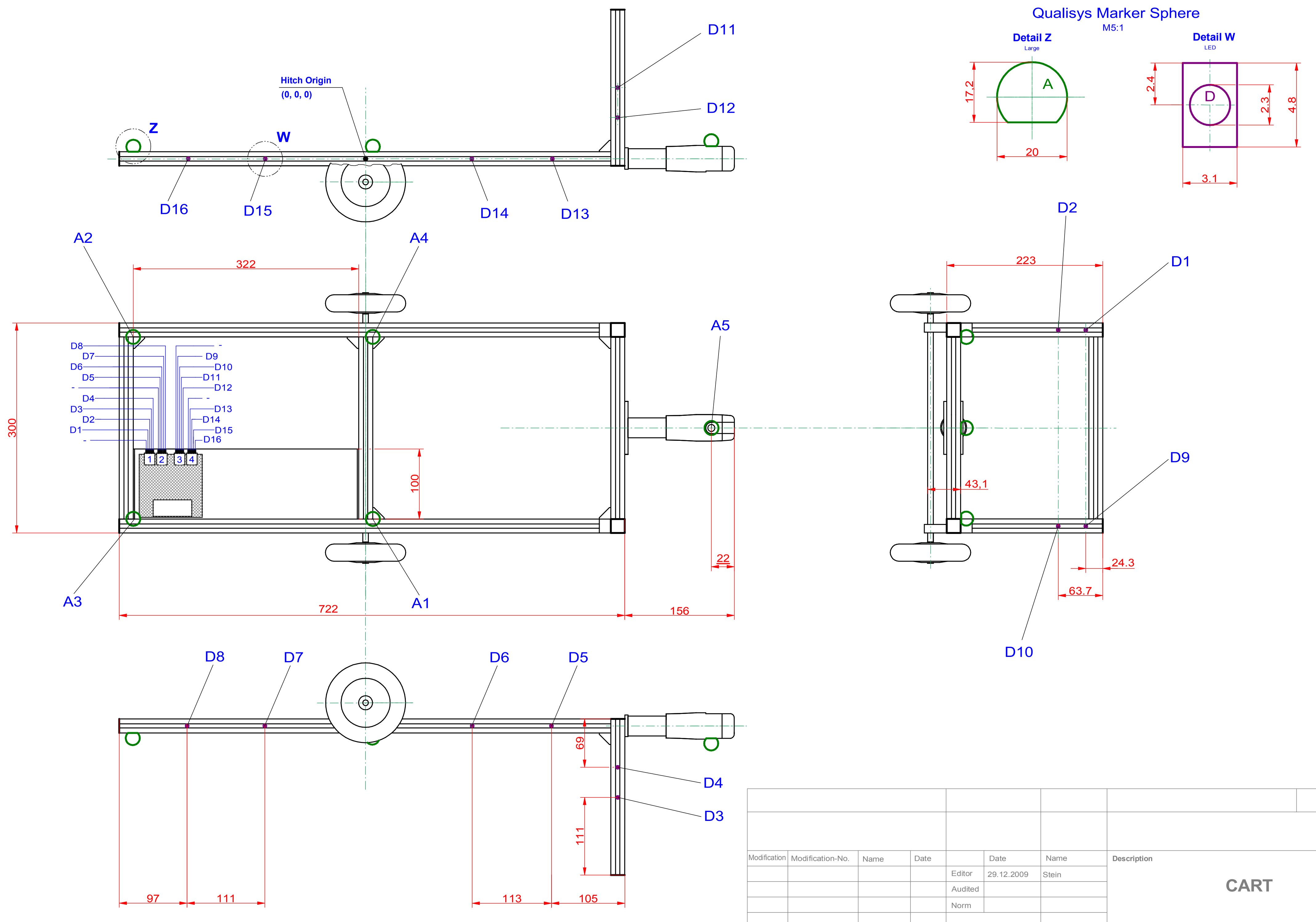
Cart To allow for reproducible revision of data from the system I included a drawing, which describes the dimension of the entire cart from the three relevant sides *back* (upper middle of the drawing), *left* (right side of the drawing like one rotates the cart along the hitch) and *right* (left). The front was omitted as there are no LEDs on this side. In the drawing each LEDs position can be checked as well as the affiliation to a certain plug and plug-group is visible.

Markers The Qualisys markers used for my tests are visible on the top left. Those markers were placed on the cart and are thus labeled as *Sphere*.

L-shaped Calibration Wand To calibrate the system I used a standard calibration tool by Qualisys. As my measurements are also dependent on its dimensions I included a drawing.

Calibration Cross The cross created of tape on the floor, which was used to match my cameras calibration with that of the motion capturing system was also included to clarify the values in my experiments understandable.

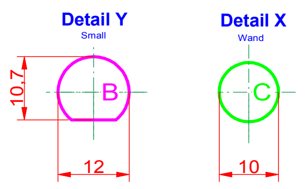
LED-Driver-Circuit The following images contain diagrams for the LED driver board used to let the LEDs blink. It was planned and constructed at the DFKI Bremen.

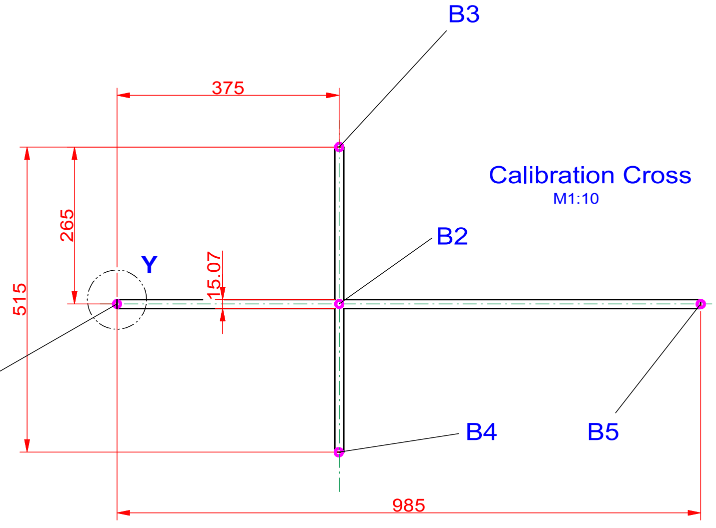


Modification	Modification-No.	Name	Date		Date	Name	Description	Scale
				Editor	29.12.2009	Stein	CART	1:5
				Audited				5:1
				Norm				
							Documentation No.	Page 1
								2 Pages

Qualisys Marker Sphere

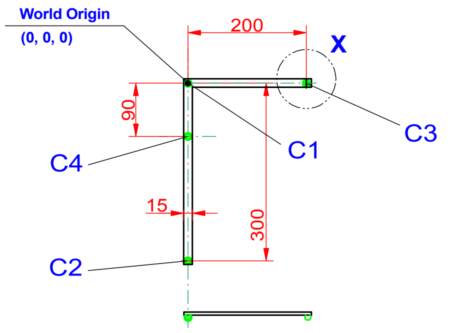
M5:1



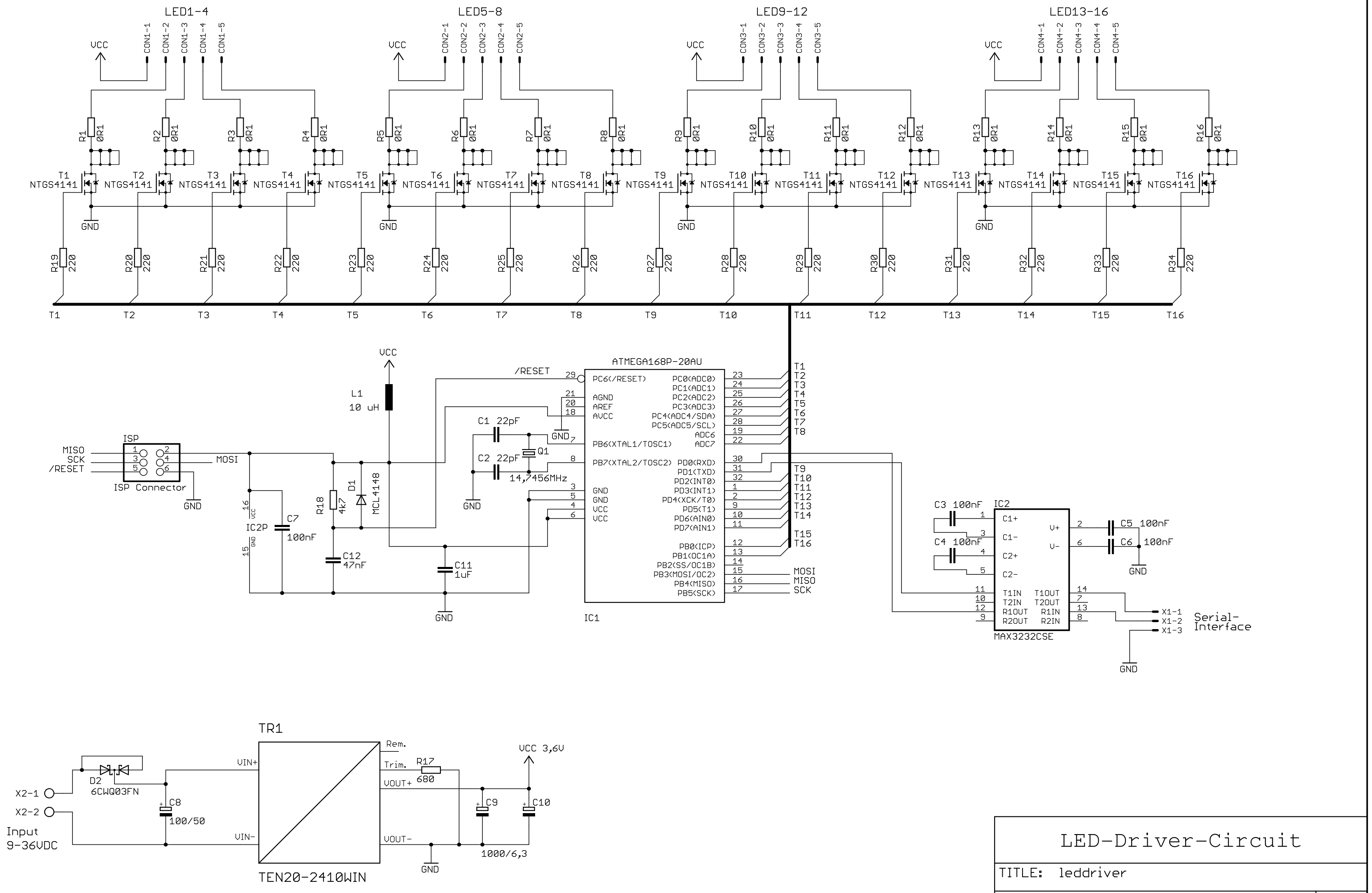


Qualisys Wand Kit 750

M1:10



Modification	Modification-No.	Name	Date		Date	Name	Description	Scale
				Editor	29.12.2009	Stein	CART Calibration	1:5 5:1
				Audited				
				Norm				
							Documentation No.	Page 2 2 Pages



LED1-4
LED5-8
LED9-12
LED13-16
VCC
CON1-1 CON1-2 CON1-3 CON1-4 CON1-5
CON2-1 CON2-2 CON2-3 CON2-4 CON2-5
CON3-1 CON3-2 CON3-3 CON3-4 CON3-5
CON4-1 CON4-2 CON4-3 CON4-4 CON4-5
R1–R16 0R1
T1–T16 NTGS4141
GND
R19–R34 220
T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15 T16
ATMEGA168P-20AU
IC1
/RESET
PC6(/RESET)
AGND
AREF
AVCC
PB6(XTAL1/TOSC1)
PB7(XTAL2/TOSC2)
PC0(ADC0) PC1(ADC1) PC2(ADC2) PC3(ADC3) PC4(ADC4/SDA) PC5(ADC5/SCL) ADC6 ADC7
PD0(RXD) PD1(TXD) PD2(INT0) PD3(INT1) PD4(XCK/T0) PD5(T1) PD6(AIN0) PD7(AIN1)
PB0(ICP) PB1(OC1A) PB2(SS/OC1B) PB3(MOSI/OC2) PB4(MISO) PB5(SCK)
MOSI
MISO
SCK
L1
10 uH
C1 22pF
C2 22pF
Q1
14,7456MHz
ISP
ISP Connector
IC2P
C7
100nF
R18
4k7
D1
MCL4148
C12
47nF
C11
1uF
IC2
MAX3232CSE
C3 100nF
C4 100nF
C5 100nF
C6 100nF
C1+ C1- C2+ C2- V+ V-
T1IN T2IN R1OUT R2OUT T1OUT T2OUT R1IN R2IN
X1-1
X1-2
X1-3
Serial-Interface
TR1
TEN20-2410WIN
VIN+
VIN-
Rem.
Trim.
VOUT+
VOUT-
R17
680
VCC 3,6V
C8
100/50
C9
C10
1000/6,3
D2
6CWQ03FN
X2-1
X2-2
Input
9-36VDC

LED-Driver-Circuit

TITLE: leddriver

Document Number:

REV:

Date: 03.12.2008 13:29:00

Sheet: 1/1