Josef Steinbäck BSc

Integration of a Time-of-Flight 3D Camera into a Mobile Sensing Platform

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Univ.-Doz. Dipl.-Ing. Dr. techn. Daniel Watzenig

Institute of Electrical Measurement and Measurement Signal Processing

Advisors: Dipl.-Ing. Dr. techn. Norbert Druml
Dipl.-Ing. Dr. techn. Allan Tengg

Graz, September 2016

Affidavit

Abstract

Automated driving has recently become an important subject that will most probably dominate the automotive industry in the next decades. Assisted driving functions have been around for quite a while, and new systems and ideas are continually being presented. There is a fair chance of deploying automated driving to the streets within the next couple of years. To accomplish that, the general public has to build trust in the systems. Automated driving systems have to operate at the highest level of robustness and safety. Thus, redundancy and diversity of different technologies is inevitable in order to guarantee the functionality in any possible scenario.

Today the most used sensor technologies for automotive environment perception are 2D cameras, radar, lidar and ultrasonic sensors. An uprising technology on this sector are Time-of-Flight (ToF) cameras, providing high frame rate 3D data with minimal computation overhead. Compared to other systems, ToF cameras are very compact and inexpensive. This work evaluates the feasibility of a 3D ToF camera in order to be used in automated/assisted driving systems. In addition, limitations and imperfections of ToF cameras in this field of application are analyzed. These include ambient light, motion artifacts and the restricted range.

To examine its performance in the field, a ToF image processing system was attached to a remote-control 1/5 scaled vehicle. Further, an algorithm was developed to compute the distance image from the sensor data and to processes the image with the purpose to detect possible obstacles. Together with the data of the scaled vehicle's current state (i.e. velocity, direction) the system estimates whether a collision is likely to occur. If so, emergency braking is initiated to avoid the crash or at least to reduce the extent of it. Due to the limited computational power of the used automotive microcontroller, a very efficient image processing algorithm is required in order to accomplish real-time performance. Utilizing all three cores of the automotive microcontroller, the system achieves frame rates of more than 30 Frames per Second (FPS).

Kurzfassung

Autonomes Fahren war und ist das Stichwort der Automobilindustrie und wird die technologischen Entwicklungen in den nächsten Jahrzehnten weiterhin wesentlich mitbestimmen. Teilautomatisierungen und Fahrerassistenzfunktionen wie die Einparkhilfe gehören bereits zum Standard, neue Systeme und Ideen werden laufend vorgestellt. Die Entwicklungen deuten darauf hin, dass automatisiertes Fahren in den kommenden Jahren mehr und mehr Einzug auf den öffentlichen Straßen findet. Dementsprechend muss die Öffentlichkeit den eingesetzten Technologien vertrauen können. Automatisierte Fahrfunktionen müssen sich daher bezüglich Zuverlässigkeit und Sicherheit auf dem höchstem Niveau bewegen. Um die Funktionalität in jedem erdenklichen Szenario zu gewährleisten, sind Redundanz und Vielfältigkeit beim Einsatz dieser Technologien unumgänglich.

Die heute am häufigsten verwendeten Sensortechnologien für die Umfelderkennung von Fahrzeugen sind 2D Kameras, Radar, Lidar und Ultraschallsensoren. Eine relativ neue und aufstrebende Technologie ist jene der Time-of-Flight (ToF) Kameras. Diese liefern 3D Bilder mit hohen Bildraten und geringem Rechenaufwand und sind im Vergleich zu anderen Systemen kompakt und kostengünstig. Diese Arbeit evaluiert die praktische Umsetzbarkeit einer 3D ToF Kamera für die Umfelderkennung in Fahrassistenzsystemen. Ein zusätzlicher Fokus liegt auf den Beschränkungen und Unzulänglichkeiten der ToF Kamera auf diesem Anwendungsgebiet, konkret wird dabei auf den Einfluss von Umgebungslicht und Bewegungsartefakten sowie den eingeschränkten Entfernungsbereich eingegangen.

Um die Anwendung des Systems in der Praxis zu testen, wurde eine ToF Verarbeitungseinheit auf einem im Maßstab 1/5 skalierten Elektrofahrzeug montiert. Damit einhergehend wurde ein Algorithmus entwickelt, der aus den Sensordaten ein Entfernungsbild errechnet und etwaige Hindernisse erkennt. Zusammen mit den Daten über den aktuellen Zustand des skalierten Fahrzeugs - etwa dessen Geschwindigkeit und Fahrtrichtung - berechnet das System die Wahrscheinlichkeit einer Kollision. Übersteigt diese einen Schwellwert, wird eine Notbremsung eingeleitet, um den Zusammenstoß zu vermeiden oder zumindest das Ausmaß zu verringern. Aufgrund der begrenzten Rechenleistung des verwendeten Mikrocontrollers ist ein sehr effizienter Bildverarbeitungsalgorithmus notwendig, um Echtzeitfähigkeit zu erreichen. Unter effizienter Nutzung aller drei Kerne des Mikrocontrollers kann das System Bildwiederholungsraten von über 30 Bildern pro Sekunde erreichen.

Acknowledgements

This thesis was carried out at the Institute of Electrical Measurement and Measurement Signal Processing at the Technical University Graz in cooperation with Infineon Technologies Austria in Graz and the Virtual Vehicle Research Center in Graz. At the beginning of this thesis I want to seize the chance to thank all the people who supported me during my studies and especially during my thesis.

Especially, I would like to express my sincere gratitude to my supervisor Univ.-Doz. Dipl.-Ing. Dr. techn. Daniel Watzenig and to both of my advisors Dipl.-Ing. Dr. techn. Norbert Druml and Dipl.-Ing. Dr. techn. Allan Tengg for their continuous support during the creation of this work, their patience and their contributed knowledge. Furthermore, I want to thank my colleagues at the "Cooperative Research and Exploration" department of the Infineon Development Center Graz for the great working atmosphere.

I also want to thank all new friends I found during my time at the university for making it such a great experience and period of my life. In particular, I want to thank my long term study-group and very good friends Christoph, Lukas and Josef for their backup and support in any situation during our years at the TU Graz.

In addition, I must express my very profound gratitude to my family and my friends for their support and patience during my studies and during the work on my thesis. I am very grateful for the unconditional support of my mother Brigitte and for her words of advice in difficult life situations. Finally, I want to thank my father Josef, the greatest man I have ever known, for inspiring me to choose a technical career path. Thank you!

Graz, September 2016 Josef Steinbäck

Contents

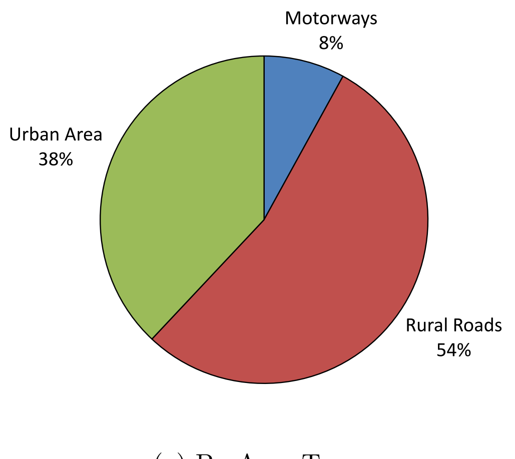
Chapter 1

Introduction

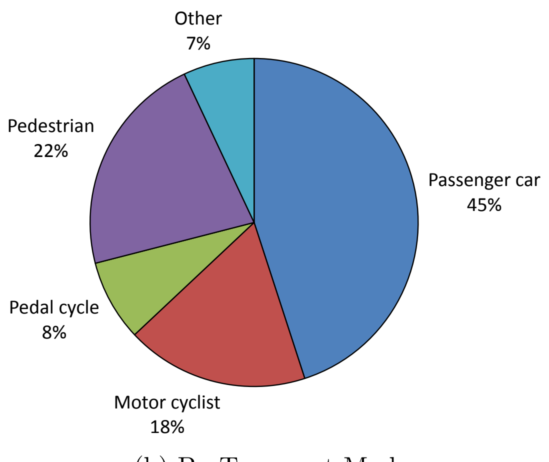
1.1 Motivation

The number of road fatalities has decreased over the last decades due to the introduction of new safety features in modern vehicles. Still, the number of car accidents caused by human failure is comparatively high, and measures are taken to decrease that number.

The Annual Accident Report of the European Union [Eur15] shows the statistics of road accidents in the European Union in 2015. The data is obtained from the Community Database on Accidents on the Roads in Europe (CARE) database. The report reveals that 38% of all lethal traffic accidents occur inside urban areas, 8% thereof affect cyclists and 22% pedestrians. The underlying statistics for road fatalities in the EU are shown in Figure 1.1. Consequentially, an enhanced view upon unprotected traffic participants and their safety is necessary.



(a) By Area Type.



(b) By Transport Mode.

Figure 1.1: Road fatalities in the EU, 2013 [Eur15].

While the protection of the car passengers has steadily improved over the last decades, the protection of pedestrians and other unprotected traffic participants has stayed more or less the same. Many accidents are caused by the fact that the driver does not notice an obstacle or cannot react fast enough. At this point modern sensor technology could be used to improve the current situation. Real-time sensor data is able to detect potential obstacles within the surrounding area of a vehicle and react to dangerous situations within milliseconds. Hence, there is a huge potential in this area of technology to make the roads safer.

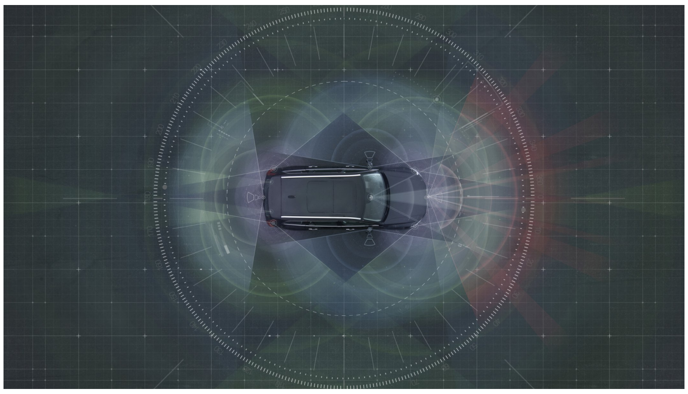


Figure 1.2: State-of-the-art surround view [Vol16].

A number of different sensors are already used in modern cars' assisted driving systems. Figure 1.2 shows the combined visible area of multiple sensors installed in modern cars. To work reliably, those assisted driving systems are in need of robust sensor data. Further, to assure the functionality in difficult and more complex situations, the safety critical systems require both redundancy and diversity. As a consequence of this there is a major interest for the automotive industry to utilize new sensor technologies and evaluate their feasibility in future systems.

The fusion of highly reliable sensor data and the ability to process this very data in real-time is essential to roll out fully autonomous driving in a safe and generally accepted way. With fully autonomous driving, traffic accidents caused by human failure could be completely wiped out.

1.2 Objectives

This thesis focuses on a 3D imaging sensor using the ToF principle. The sensor's feasibility is evaluated for the front-view use on a vehicle. The main goals of this work are:

- Build up a system consisting of a 3D ToF camera and an automotive microcontroller.
- Install this system on a mobile platform (a 1/5 scaled vehicle).
- Make the sensor's 3D images available to the microcontroller.
- Develop a fast algorithm to process the 3D data, meeting real-time constraints.
- Detect obstacles and stop the moving vehicle in situations when a collision is likely to occur.

- Estimate the usability of the 3D camera for front-view application on a full-sized passenger car.

The construction of this system will evaluate if the 3D camera is a feasible application for environmental perception. If so, the use of a 3D camera could significantly improve the quality of surround detection of the vehicle's environment and thus lead to better driver assistance or autonomous driving performance. Another possible use case is installing the camera as an additional system for surround detection to add diversity and redundancy to the system. The ToF object detection might work in special situations when other sensors fail, for example, in bad lighting conditions, where regular 2D cameras provide a low image quality.

1.3 Outline

Chapter 2 contains the theoretical part of this work. The current state-of-the-art of surround-view sensors in modern cars is described. The principle of ToF cameras, their imperfections and current applications is introduced. In addition, different methods of achieving object detection with 3D data are explained. A requirement analysis to point out the single requirements of the thesis is performed in Chapter 3. Additionally, the used components of the system are described in detail and the structure of the processing algorithm is introduced. Chapter 4 explains the workflow of the actual development as well as implementation related details of both hardware and software. The results of the thesis are presented in Chapter 5. The built-up platform is described and the performance of the system is evaluated in terms of quality, robustness and speed. Finally, Chapter 6 summarizes the results and gives suggestions for possible future work based on this thesis.

Chapter 2

Literature

This chapter gives an overview of the current state-of-the-art of the problem targeted in this thesis. A rough overview of automated driving and the currently used sensor technologies is given. Afterwards, the ToF principle is described and different methods of performing object detection are introduced.

2.1 Automated Driving

Automated driving recently became an important subject and there is a lot of investment in improving the technology. This section first introduces the different levels of automated driving and then focuses on 3D sensor data.

2.1.1 Levels of Automated Driving

In automated driving, there exist different classifications for automated driving systems from different institutions. Standardization is important to help customers, politicians and the industry to distinguish between the available assisted/automated driving functionalities.

The Society of Automotive Engineers (SAE) has published the currently most acknowledged standard, defining six different levels of driving automation for on-road vehicles [Sae14]. A rough overview of the different levels is illustrated in Figure 2.1. Currently the car industry offers cars with built-in assisted driving functions up to level 2. The introduction of higher-level systems also depends on the progress done in legislation and politics.

3D ToF cameras can be used in systems of different levels. In lower level systems (level 0 or 1), a ToF camera can be used to obtain environment data and implement functions like emergency breaking or reverse driving assistance. In higher level systems (level 2, 3 or 4) the main field of application is interior monitoring. In situations when the assisted/automated driving system gives the control back to the driver, a ToF camera can be used to detect if there is a person ready to take over at the driver's seat.

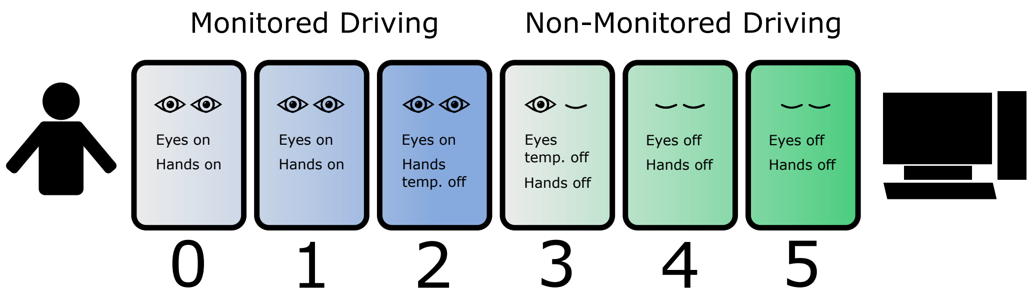


Figure 2.1: Levels of automated driving.

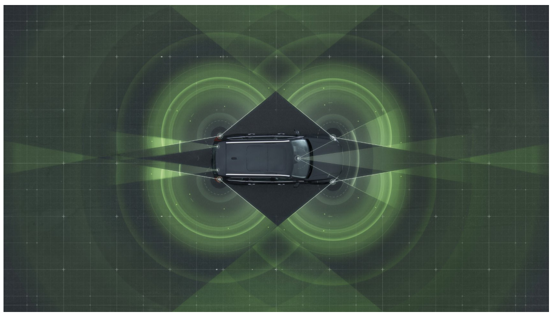
The levels of automated driving are introduced here with examples for each level:

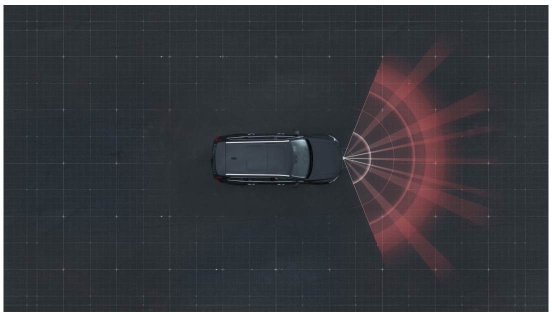
- **Level 0 (no automation):**
  The human driver has to perform all actions on their own. The system does not take over control and at most can only warn the driver. E.g.: lane change assist, park distance control, lane departure warning, front crash warning.
- **Level 1 (driver assistance):**
  The system can either perform steering or acceleration/braking. The human driver has to operate the remaining tasks and at any time be ready to take over full control. E.g.: adaptive cruise control, park assistance (level 1), lane keeping assist, front crash prevention.
- **Level 2 (partial automation):**
  The system can perform steering and acceleration/braking. The human driver has to be ready to take over full control at any time. E.g.: park assistance (level 2), traffic jam assist.
- **Level 3 (conditional automation):**
  The system can perform steering and acceleration for given use cases. It can request the driver to intervene, when it recognizes its limits. The human driver does not always have to monitor the driving, but shall be able to resume control within a certain time. E.g.: traffic jam chauffeur (level 3), highway chauffeur (level 3).
- **Level 4 (high automation):**
  The system is capable of driving on its own for given use cases. Within those use cases, the driver does not need to monitor the system or take control. E.g.: parking garage pilot, traffic jam chauffeur (level 4), highway chauffeur (level 4).
- **Level 5 (full automation):**
  The system can perform the entire driving task in any use case. No actions from a human driver are required. E.g.: fully automated vehicle.

2.1.2 Sensors

Currently, the combination of multiple sensor technologies is used to offer assisted/autonomous driving functionality [Win09], [Vol16]. The Figure 2.2 shows the impact area of the different sensors used in the autonomous driving project “Drive Me” by Volvo [Vol13].

(a) Ultrasonic sensors.


(b) Surround radar.


(c) Multiple beam laser scanners.

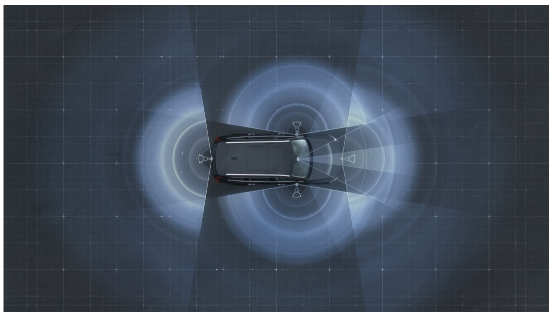

(d) Surround vision.

Figure 2.2: Volvo's autonomous driving sensors [Vol16].

- **Ultrasonic sensors:**
  Multiple ultrasonic sensors are mounted around the car to detect close objects. The combination of the data from different sensors makes it possible to increase the detection accuracy of an object. Currently those sensors are mostly used for parking assistance, but they could also be used for the detection of pedestrians that are close or other objects. The impact area of multiple ultrasonic sensors can be seen in Figure 2.2a.
- **Radar:**
  The term radar comes from radio detection and ranging. Long-range radar systems are mounted in the front to detect other road users or objects on the road. Long-range radar systems pointing backwards can be used to detect vehicles behind the car. With the mid- and short-range radars mounted on the vehicle's sides, objects around the car can be detected. The impact area of the radars is shown in Figure 2.2b.
- **Lidar:**
  Lidar is an acronym for light detection and ranging. The advantage of the laser scanners is the high angle resolution combined with a high range of more than 100 m. With multiple beams it is possible to detect objects within a wide field of view in front of the vehicle (see Figure 2.2c).
- **Cameras:**
  2D cameras have been around for many years. They are technically highly developed compared to other technologies and are inexpensive due to the availability on the

mass-market. Since the cameras strongly depend on the lighting conditions, their performance decreases at night and bad weather. Multiple cameras are placed around the vehicle to achieve a surround view (see Figure 2.2d). A special trifocal camera is placed in the front direction, combining three different cameras with different fields of view. By comparing the different images, it is possible to estimate a depth knowledge of the scene.

2.1.3 Range Imaging

3D range cameras are becoming more and more popular for use cases in automotive environments. In many situations, object detection using distance data leads to better results than with the corresponding 2D images. Since redundancy is an important topic in assisted driving and even more important in autonomous driving, 3D cameras can be used as an additional independent system along with other sensors. Other approaches combine the high resolution 2D data with the distance data from a 3D sensor [Pla16]. This can significantly improve the expressiveness of the data.

The automotive applications of 3D cameras can be split into interior monitoring and outside monitoring. Interior monitoring describes the inside use of a 3D camera. For example, for gesture control or the observation of the driver's state of fatigue. Outside monitoring focuses on the outside use of range cameras to capture the surroundings of the vehicle. Examples are a camera on the rear-side of the car for back-driving assistance or in front-direction, and around the car to detect obstacles [Win09].

In general, there are three different methods to obtain a 3D image of a scene:

- **Stereo vision:**
  Stereo vision is an approach using two cameras with a defined position. With the same point detected on both images it is possible to calculate the distance using triangulation. The downside is that finding the same point in two images is computationally expensive (solving the correspondence problem).
- **Structured light:**
  Structured light is actively emitted to the scene in a special pattern. The pattern can be detected with the camera and triangulation can be applied to gain distance information.
- **Time-of-Flight:**
  In the ToF approach, the travel time of the light is measured. An active emitter sends out a signal, which gets reflected by the scene and detected by the sensor. ToF cameras implement Photonic Mixing Device (PMD) pixels to directly measure the distance in every pixel. Thus, ToF cameras do not require a high computational effort to obtain the distance image. Additionally, ToF cameras are compact, affordable and allow high frame rates. That is why the use of ToF is very promising in dynamic automotive environments.

ToF is an uprising technology, but as it is pretty new to the industry, not many profound studies are available, which evaluate the real value of that technology in automotive environments. Most approaches in literature have focused on the interior use of ToF in cars.

Therefore, this thesis targets the feasibility of this technology for vehicle environment perception.

2.2 Time-of-Flight Imaging

This section describes the working principle of ToF cameras, introduces the most important characteristics and presents some applications.

2.2.1 Basic Principle

As the name of the technology indicates, the travel time of the light is measured. The most straightforward way to do this is to transmit a light pulse and measure the time until the reflected pulse is detected. Knowing the speed of light, the travel time corresponds to twice the distance from the object. Using continuously modulated infrared light instead of a light pulse simplifies the measurement process in the hardware. Instead of direct measurement of the time, the phase difference is measured. An illumination unit transmits the light and a PMD sensor detects the reflected light to determine the phase difference efficiently. To obtain the depth information of the whole scene instead of a single pixel, an array of PMD pixels behind an optic lens is used [Lan00]. Figure 2.3 shows the basic ToF principle, with the use of modulated light and a PMD array.

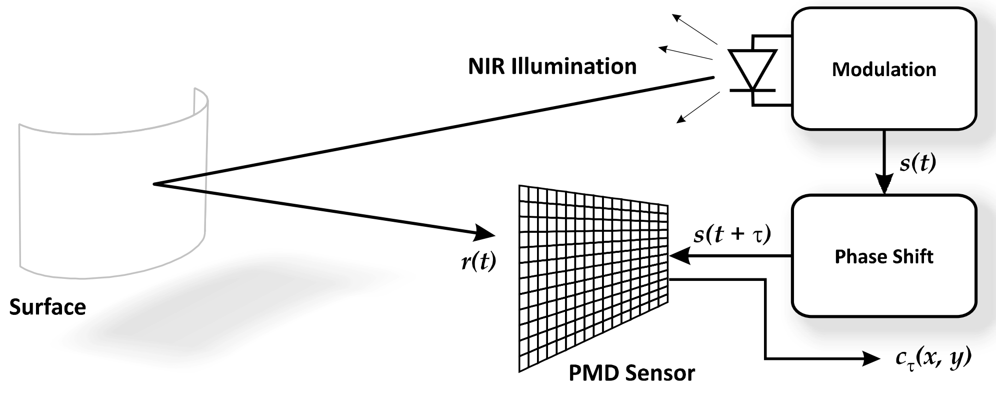


Figure 2.3: Illustration of the ToF principle [Lin10].

Photonic Mixing Device

The PMD sensor can directly measure the cross correlation value $c_\tau(x, y)$ of the received signal $r(t)$ and a phase shifted version $s(t + \tau)$ of the transmitted signal $s(t)$. Figure 2.4 shows the structure of a PMD sensor in Complementary Metal–Oxide–Semiconductor (CMOS) technology. Caused by the photoelectric effect, incoming photons translate to electron-hole pairs in the substrate. The modulated reference signal $s(t + \tau)$ is applied to the photosensitive photogates in order to create a potential variation within the substrate.

Depending on the phase difference between the received signal and the reference signal, the photoelectrons drift either to the left or the right diode. After a certain illumination time, the difference between the two voltages U_A and U_B is read out, expressing the correlation of the two signals.

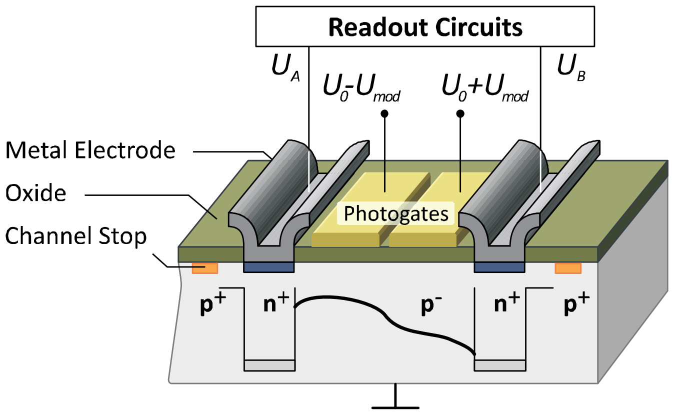


Figure 2.4: PMD sensor layout [Lin10] and [Möl05].

Amplitude and Distance Calculation

The cross correlation value C is typically determined with four different phase shifted versions of the transmitted signal $s(t+\tau)$ for $\tau = \{0°, 90°, 180°, 270°\}$. With those four cross correlation values it is possible to calculate the intensity value A with [Lan01]:

$$A = \frac{\sqrt{(C_{270°} - C_{90°})^2 + (C_{0°} - C_{180°})^2}}{2} \tag{2.1}$$

The phase difference $\Delta\varphi$ can be determined with [Lan01]:

$$\Delta\varphi = \arctan\left(\frac{C_{270°} - C_{90°}}{C_{0°} - C_{180°}}\right) \tag{2.2}$$

The distance d to the reflected object can be determined using the speed of light $c_0 \approx 3 \cdot 10^8 \frac{\mathrm{m}}{\mathrm{s}}$, the modulation frequency f_{mod} and the phase difference $\Delta\varphi$ (see Equation 2.3). Since the light travels back and forth until it is detected, the whole term has to be divided by two.

$$d = \frac{1}{2} \cdot \frac{c_0}{f_{mod}} \cdot \frac{\Delta\varphi}{2\pi} \tag{2.3}$$

The main computational expense to obtain the distance image from the sensor's four raw phase-images is the arctangent calculation. The choice of the best arctangent algorithm strongly depends on the targeted architecture and the use case. There exist mainly three different approaches to implement the inverse tangent function in embedded systems:

- **Series expansion:**
  The use of series expansion (e.g., Taylor series, Chebyshev polynomials) can be very accurate, at the cost of high computational effort. On many microcontrollers, trigonometric functions based on series expansion are already included in the software libraries.
- **Coordinate Rotation Digital Computer (CORDIC):**
  The CORDIC algorithm is an efficient, iterative algorithm to calculate trigonometric functions [Vol59]. The algorithm can be implemented using only simple shifts and additions, but the iterative behavior limits the speed if certain accuracy is required. Using parallelism can drastically increase the throughput, making it popular to be implemented in hardware [Bel00].
- **Look-up Table (LUT):**
  A LUT holds the arctangent values for the required input range. The accuracy can be further improved by using linear interpolation. This approach is very fast but requires a certain amount of memory. The work in [Uki11] shows an implementation of this approach for embedded systems with very limited computational resources.

2.2.2 Characteristics

The output of a ToF camera depends on various parameters and properties. The most influential characteristics are introduced and discussed here.

Modulation Frequency

The modulation frequency determines the maximum range of a single ToF measurement. For the unambiguous range, the phase difference $\Delta\varphi$ has to be within 0 and 2π. This limits the maximum unambiguous distance $d_{u,max}$ to:

$$d_{u,max} = \frac{1}{2} \cdot \frac{c_0}{f_{mod}} \tag{2.4}$$

A lower frequency results in a higher unambiguous range, but lower accuracy within that range. This unambiguous distance shall not be confused with the range of a ToF camera. Objects beyond the maximum unambiguous distance might still be detected but result in a wrong distance value.

Illumination Time

The illumination time directly affects the quality of the obtained image. It is equivalent to the integration time of the photons on the PMD sensor. A higher illumination time comes with important advantages. It increases the signal-to-noise ratio (SNR) and the robustness of the measurement. But on the other hand, it can lead to overexposure of close and highly reflective objects. Additionally, the vulnerability to motion blur and motion artifacts increases, as well as the power consumption.

Range

The effective range of the ToF camera is not only limited by the modulation frequency but also by the transmitted signal strength. Some objects exceeding a certain distance are not reflected strongly enough to be detected properly, while others at the same distance still result in a solid distance value.

For a given transmission signal strength, the amplitude and thus the quality of the distance data of one pixel depends on the albedo ρ, the distance to the reflecting object r and the angle θ to its surface [Mur07]. This reflection is shown in Figure 2.5 and described with the following formula:

$$A \propto \frac{\rho \cdot cos(\theta)}{r^2} \tag{2.5}$$

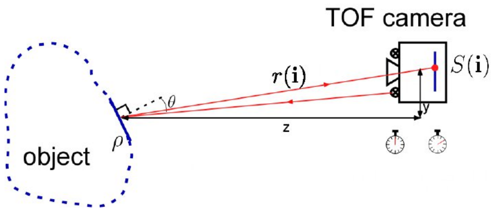


Figure 2.5: Reflection characteristics of the emitted light [Mur07].

The most straightforward way to increase the range would be to increase the transmitting strength of the infrared signal. But in many application fields this is only allowed up until a certain magnitude due to eye-safety regulations. This causes the usable range of current ToF cameras for automotive use to be limited to a distance of less than 10 m.

Accuracy

The accuracy of a ToF camera depends on multiple factors (e.g., modulation frequency, noise and systematic errors). The compensation of systematic errors is mandatory to achieve high distance accuracy. The data required for the compensation can be obtained using calibration. Systematic errors include: the global offset, fixed pattern noise, temperature dependent error, distance dependent error, etc. But there are also random errors present, which are not able to be compensated, and limit the maximum accuracy of the system. Those random errors include light scattering, multi path reflections, shot noise and other quantification effects. Additionally, the measurement accuracy increases with an increasing modulation frequency. After the compensation of the systematic errors, current ToF systems can achieve precisions in the area of a few millimeters [Win09].

The calculation of the distance data using finite numbers and approximations will also add an uncertainty. This error has to be considered, especially when embedded systems with fixed point arithmetic are used.

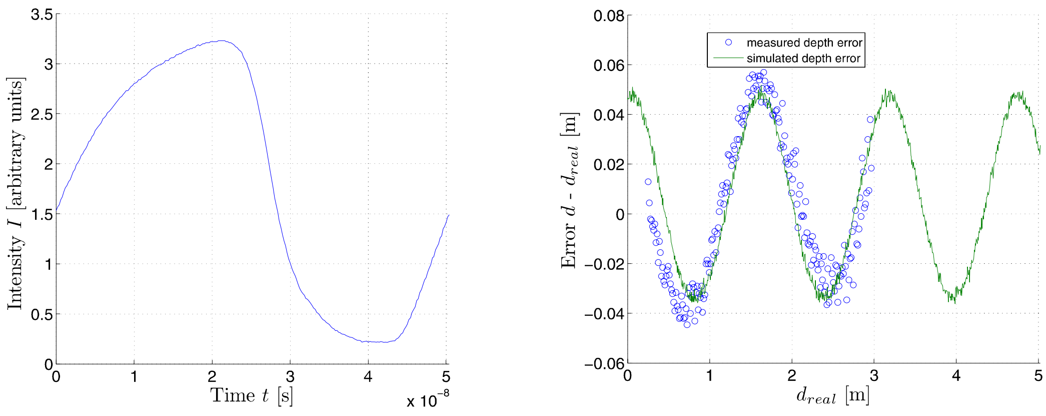
Resolution

The resolution of most current ToF cameras is within a range of 160 x 120 pixels and 352 x 288 pixels. Compared to current 2D cameras with multiple megapixels, the resolution is very low and might not be sufficient for certain applications with the need of fine-grained data.

Drawbacks

This subsection describes the imperfectness of the ToF measurement with the use of a PMD sensor. The most relevant errors that occur during the measurement are described here and their impact on the output is explained. Some errors can not be compensated. Some compensation methods are described as well.

- **Wiggling error:**
  The wiggling error is a distance dependent error and is caused by the imperfection of the sinusoidal modulation. The distance equations only lead to exact values if the modulation signal is a perfectly shaped sine. In reality the modulation signal is rather a digital rectangle signal (see Figure 2.6a) and thus the distance equations lead to an imprecise result. As seen in Figure 2.6b, the distance error caused by the wiggling effect follows a sinusoidal shape. There exist different approaches to compensate the wiggling error. One is to determine the distance error for a sufficient amount of distance values using an exact reference and compensate it using a LUT. Another possibility is to compensate the error using a mathematical model, derived from the sinusoidal shape of the error.



(a) Modulation signal. (b) Wiggling error.

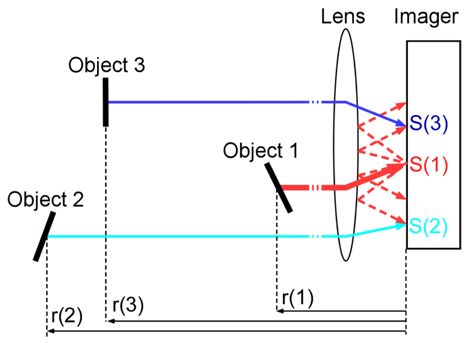
Figure 2.6: Wiggling Error [Sch09].

- **Ambient light:**
  An optical spectral filter is applied to the sensor to block all light waves outside the relevant infrared range. If ambient light is present at the same wavelength as the emitted signal, it causes the image quality to decrease. Although it is usually not modulated, it can still cause the sensor's pixels to enter saturation. A suppression of

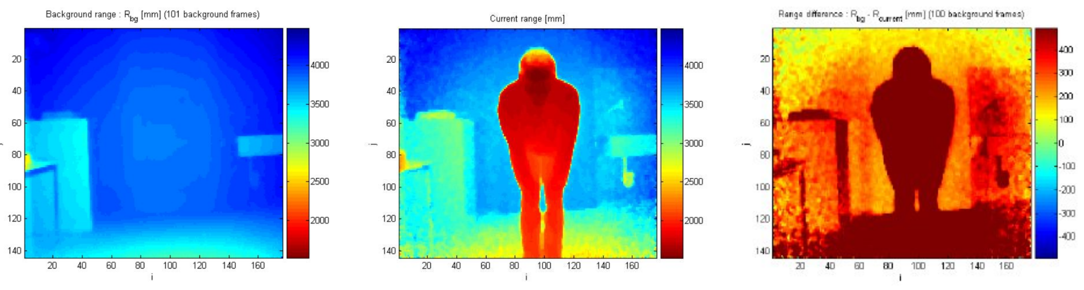
background illumination can be applied using special compensation circuits, rejecting ambient light up to a certain magnitude and thus avoiding saturation [Möl05]. Additionally, ambient light causes photon shot noise on the sensor [Gok04].

- **Overexposure:**
  Highly-reflective objects close to the lens can cause overexposure of pixels. The PMD sensor has a fixed capacity and only works properly up until a certain amount of photons are received. The impact can be reduced by lowering the exposure time, this may cause however that badly reflective objects at a certain distance to not be detected anymore. So there is always a trade-off between risking overexposure and not detecting objects.

  A related error to overexposure is called light scattering (stray light inside optics), describing the reflections between the lens and the image sensor [Mur07]. It occurs because the infrared light is scattered over the whole image and causes the background to appear at a closer distance. Figure 2.7 illustrates the origin of the effect and shows the influence of light-scattering on the distance image. The impact of this error significantly increases when overexposure occurs and also causes many of the remaining image pixels to become unusable.



(a) Light scattering, principle.



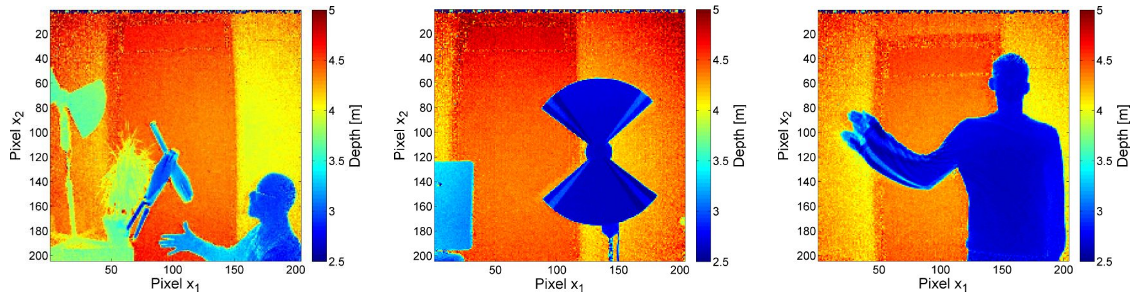
(b) Only background. (c) With foreground. (d) Distance difference.

Figure 2.7: Impact of light scattering [Mur07].

- **Motion artifacts:**
  In dynamic scenes with moving objects or a moving camera, motion artifacts are very likely to occur. Motion artefacts shall not be confused with motion blur. The latter concerns the distance uncertainty that occurs by motion during the integration

time. Since the integration time is usually very low in ToF systems, the influence of motion blur can be neglected in many cases. The major problem is motion artifacts that describe the distance error caused by changes of the scene during the sequential measurement of the four phase-images. Figure 2.8 shows a few examples of motion artifacts in distance images.

In certain applications, reducing the illumination time can be sufficient to cope with motion artifacts in dynamic scenes. In other applications it might be mandatory to apply certain algorithms to identify and/or correct motion artifacts within the distance data. Examples for such algorithms are presented in [Hoe13], [Sch15] and [Sch11a].



(a) Juggling. (b) Rotating fan. (c) Fast arm movement.

Figure 2.8: Distance images with motion artifacts [Sch11b].

- **Phase-wrapping:**
  From the raw ToF data of a single measurement it is not possible to distinguish between $\Delta\varphi = \Delta\varphi + n \cdot 2\pi$ phase data. This effect leads to wrong distances for pixels with corresponding distances further away than the unambiguous range. There exist some phase unwrapping approaches to extend the unambiguous range of the ToF camera [Han13]. One method is to execute sequential measurements with different modulation frequencies that are combined to extend the unambiguous range of the distance data [Gok04]. The disadvantage is the additional delay, introduced by the increased capturing and processing time. This additionally limits the frame rate and can cause motion artifacts in dynamic scenes.

2.2.3 Applications

Many new applications arise with the ability to capture 3D data at a high frame rate.

Human-Machine Interaction

There exist multiple applications where ToF cameras are used for human-machine interaction. Examples are gesture control (e.g., of media devices in a car) and augmented reality.

A well-known example for a ToF camera used for human machine interaction is the Kinect for Microsoft's Xbox One. The Kinect is a user interface to use motions of players as input to the gaming console to enhance the gaming experience. Compared to other currently available sensors, it offers a high resolution of 512 x 424 pixels [Bam15].

With ToF cameras real-time motion tracking can be achieved marker-less. A single ToF camera can be used instead of expensive and complex camera systems. Approaches to accomplish real-time motion capture and human 3D pose estimation are presented in [Gan10], [Pla10] and [Sch11c]. The Figure 2.9 shows a full-body pose estimation using a ToF camera.

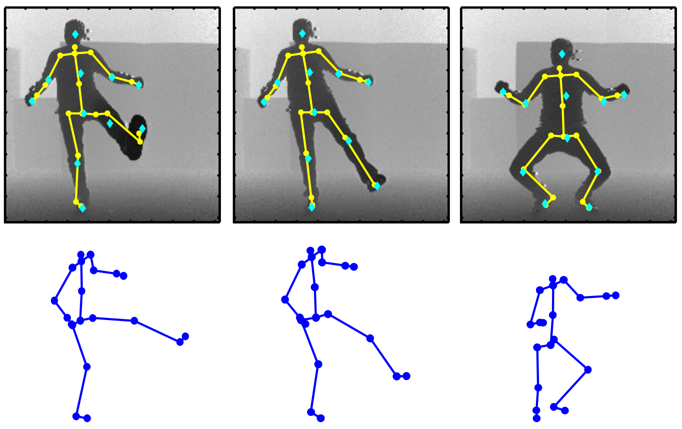


Figure 2.9: Full-body pose estimation. The distance images are overlaid with the estimated skeleton pose. Obtained from [Sch11c] with changes.

Automotive Assistance Functions

The authors in [Sch07] evaluated a special PMD camera for automotive use with the result that the ToF principle can be used for obstacle detection and tracking (e.g., for pre-crash detection).

The work in [Dal14] presents an approach with four ToF cameras mounted on a vehicle to gain a 360 degree view of the vehicle's surrounding area. Static and dynamic objects around the vehicle are detected in real-world conditions. Figure 2.10 shows the field of view of the single cameras mounted on different sides of the vehicle.

In [Wei14] a ToF vision system is proposed that is able to perform pedestrian detection. First the scene is efficiently segmented using the distance histogram and then the range values are clustered using a mean-shift algorithm. Afterwards Fourier and Generalized Search Tree (GiST) features are extracted for each region. Finally, the segmented regions are classified into pedestrians and non-pedestrians using a Support Vector Machine (SVM) as classifier.

Face Detection

The availability of distance data can drastically improve the quality of face detection. Due to the distance information, it is possible to get additional information, for example

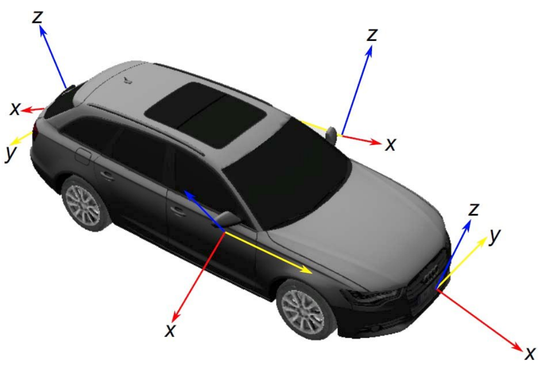


Figure 2.10: Full surround view with four ToF cameras [Dal14].

about the position of a human head within the image. The works presented in [Böh09] and [Fis10] show the value of a range measurement in addition to a regular color image in order to improve the quality and the speed of face detection.

The work in [Wal07] presents a simple way to implement person counting using a PMD based ToF camera. A ToF camera was placed on top of a door facing the ground, and the resulting distance images were scanned for human heads.

2.2.4 REAL3 Image Sensors

Infineon Technologies AG and PMD Technologies AG started a cooperation to develop a 3D ToF image sensor. A first generation of 3D image sensors was released in 2013 [Inf13]. The sensor was advertised as the most integrated and sophisticated ToF imager available on the market. Two products with different resolutions were introduced (see Table 2.1). A second generation was presented in 2015 [Inf15]. The new sensors have a highly improved sensitivity due to the use of special micro-lenses. Additionally, there exist versions with smaller size and lower resolution for low-power applications like the use in mobile devices. An overview of the different products can be seen in Table 2.1 and Figure 2.11. PMD Technologies offers the CamBoard pico flexx, a reference camera with the ToF image sensor (see Figure 2.12). The small USB device comes with a Software Development Kit (SDK), which makes it possible to use the provided sensor data in custom applications.

	Product Name	Resolution
Generation 1	IRS1010C	160 x 120
	IRS1020C	352 x 288
Generation 2	IRS1125C	352 x 288
	IRS1645C	224 x 172
	IRS1615C	160 x 120

Table 2.1: Released image sensors from Infineon Technologies in cooperation with PMD Technologies.

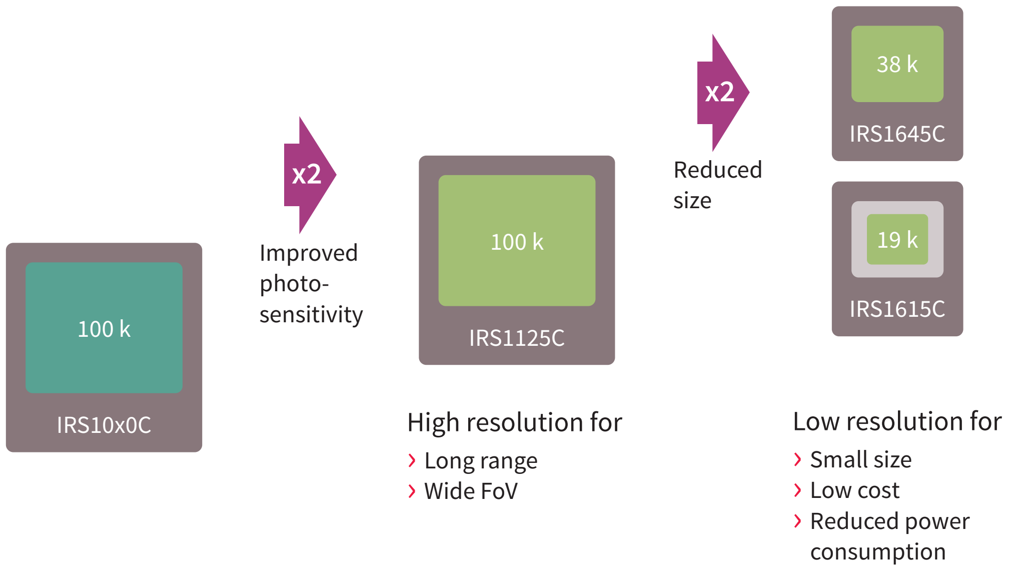


Figure 2.11: Product evolution of the ToF image sensors [Inf15].

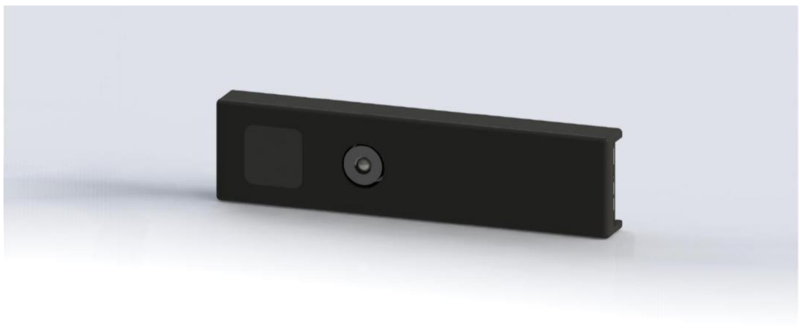


Figure 2.12: CamBoard pico flexx reference design [Pmd15].

2.3 Object Detection

This section describes the image processing steps to achieve object detection with 3D images. There exist various different approaches for 2D images, but since 3D cameras are relatively new to the industry, there are not so many well-founded 3D object detection principles. Some parts of the principles can be easily modified for 3D cameras while others are not applicable for distance data.

For use in an automotive environment, the system shall detect objects in front of the ToF camera, isolate them and get the area, position and distance of each object. The detected objects should be re-recognized in the following frames to assign trajectories to every object, representing their relative movement to the ToF camera. Considering the car's speed, direction, etc., it is possible to decide for each object independently if intervention is required.

2.3.1 Pre-Processing

For pre-processing of image data, different filters can be used. The choice of the filter depends on the type of the image itself, and under what circumstances it was taken. In ToF 3D images, there is usually some noise in the image. This holds especially true for pixels, where there is no object present to reflect the emitted light.

Median Filter

A median filter can be used to omit single erroneous pixels and thus decrease the noise of the image. The 8-neighborhood is considered and the processed pixel is set to the median value. This can be implemented time-efficient and leads to good results. The filter has the advantage, that it preserves edges and does not blur them.

$$I_{out}(x,y) = \underset{(x,y)\in 8N}{\text{median}}\{I_{in}(x,y)\} \tag{2.6}$$

Mean Filter

Another smoothing technique is to apply a mean filter. This filter considers the neighboring pixel values, usually of the 8-neighborhood, and sets the processed pixel to the mean value. The mean filter smooths the whole image, but sharp edges get lost since the whole image gets blurred. Also single pixels with a totally wrong distance value can affect the output significantly.

$$I_{out}(x,y) = \frac{1}{9} \sum_{(x,y)\in 8N} I_{in}(x,y) \tag{2.7}$$

Common Neighborhood Filter

A Common Neighborhood (CN) filter is used to detect and discard erroneous pixels [Far06], [Dal14]. The filter considers neighboring pixels in the same value-range as the currently processed pixel as common neighbors (see Equation 2.9). Usually the 8-neighborhood is taken into account. If the common neighbors exceed a certain amount, the pixel is marked as valid and is considered for further processing. Otherwise the pixel is discarded and not further processed (see Equation 2.8).

$$I(x,y) = \begin{cases} I(x,y) & \text{, if } \sum\limits_{(i,j)\in 8N} h(i,j) > T_1, \\ \text{invalid} & \text{, otherwise.} \end{cases} \tag{2.8}$$

$$h(i,j) = \begin{cases} 1 & \text{, if } |I(i,j) - I(x,y)| \leq T_2, \\ 0 & \text{, otherwise.} \end{cases} \tag{2.9}$$

Amplitude Thresholding

Since the raw sensor data can be used to obtain amplitude data in addition to the distance data, the amplitude values can be used to make decisions about the quality of the distance image. Distance pixels with low amplitude values are very likely to be erroneous and can be discarded. An easy approach to achieve this is shown in Equation 2.10.

$$I(x,y) = \begin{cases} I(x,y) & \text{, if } A(x,y) \geq A_{treshold}, \\ invalid & \text{, otherwise.} \end{cases} \tag{2.10}$$

Caused by the lens and the illumination unit, the reflected light is not equally distributed throughout the pixel array. The amplitude value in the center of the pixel array is higher than in outer regions for objects of the same reflectivity. To overcome this issue, an amplitude reference image can be used to define the threshold value depending on each pixel's location.

Bilateral Filtering

Bilateral filtering determines the value of the processed pixel by calculating the weighted average of the neighboring pixels. The filter achieves smoothing while preserving edges. Other than regular Gaussian smoothing, the weights for bilateral filtering do not only depend on the pixel-distance, but also on a second domain. The typical example for the second domain is the intensity values. The bilateral filter is defined as:

$$I(\boldsymbol{x_0}) = \frac{1}{W_p} \sum_{\boldsymbol{x} \in \Omega} G_s(\|\boldsymbol{x_0} - \boldsymbol{x}\|) G_i(\|\boldsymbol{x_0} - \boldsymbol{x}\|) I(\boldsymbol{x}) \tag{2.11}$$

Where W_p is the sum of all weights to normalize the result. Ω is the window around the currently processed pixel $\boldsymbol{x_0}$. G_s and G_i are the Gaussian kernels for the spatial and the intensity domain.

For ToF data, the second domain can be the z-values, while the first domain is the pixel position within the image. The filter is called a cross-bilateral filter if there is external data used for the second domain. An example for a cross bilateral filter is the use of the amplitude data for the second domain. In [Len11] a special bilateral filter for ToF data is used that combines the amplitude and the distance data for the second domain. This filter especially preserves edges seen in both, the amplitude and the distance image. Bilateral filtering leads to good results and has already been successfully applied in ToF pre-processing applications. Figure 2.13b shows an example for ToF image pre-processing using a bilateral filter. The downside is that the required calculations are significantly more computational expensive than for the previously mentioned methods.

Total Variation De-noising

In the total variation (TV) de-noising approach, the total variation of the data is reduced while the output data has to be as similar as possible to the input data. After applying this method the unwanted noise is removed while the relevant information, such as edges, is kept.

To perform TV de-noising the following objective function has to be minimized [Len13]:

$$min \left[\left(\sum_{\boldsymbol{x} \in N} \frac{1}{2} (I_{out}(\boldsymbol{x}) - I_{in}(\boldsymbol{x})^2 \right) + \lambda \cdot R(I_{out}) \right] \tag{2.12}$$

The first term assures that the output image is close to the input image, while the second is a regularization term to minimize the total variation. Figure 2.13c shows an example for ToF image pre-processing using TV de-noising. This de-noising approach leads to very good results, but solving the optimization problem is expensive in terms of computation.

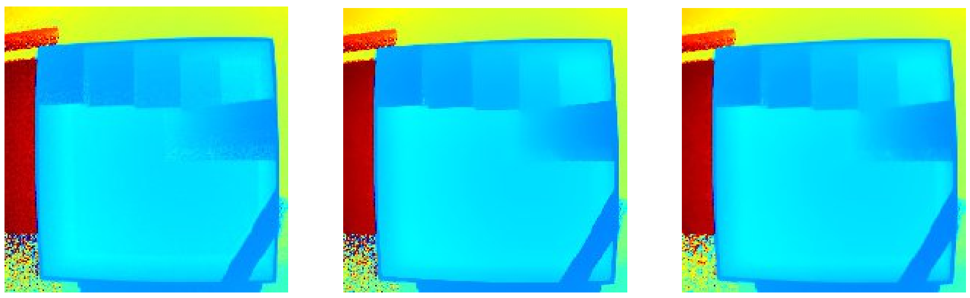

(a) Input. (b) Bilateral filter. (c) TV de-noising.

Figure 2.13: Bilateral filtering and TV de-noising [Len13].

2.3.2 Segmentation

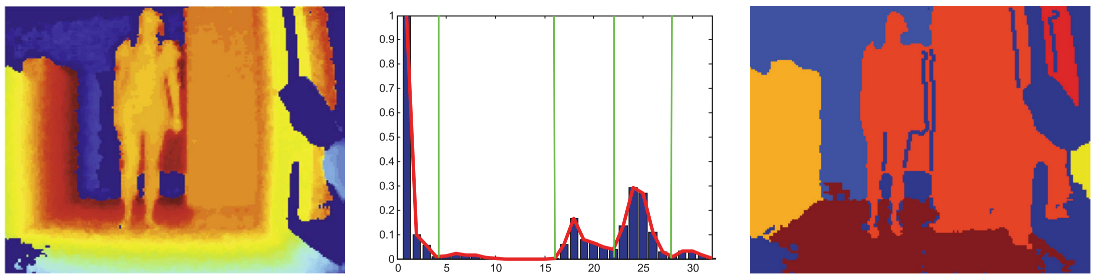
After pre-processing, the next step is to segment the distance image into different regions. A few relevant segmentation methods that are applicable to 3D ToF images are introduced here.

Distance Thresholding

A very simple and time efficient method to achieve segmentation is to apply thresholding to the image. With that approach it is for example possible to separate the foreground and the background of the image. It is also possible to extract a custom Area of Interest (AOI), by setting a threshold depending on the pixel-position.

Histogram-Based Segmentation

An efficient approach presented in [Wei14] is to achieve segmentation of a 3D image using the histogram. Figure 2.14 shows the already pre-processed input image, the corresponding histogram with the segmentation borders marked and the segmented output image with different colors for each segment.



(a) Input image. (b) Histogram. (c) Segmented image.

Figure 2.14: Histogram-based segmentation [Wei14].

First the histogram of the pre-processed image has to be created, dividing all distance values into histogram-bins with a certain range. In order to detect the local minima of the histogram, the first-order derivative has to be determined. For a histogram h with N bins, the first-order derivative d_n has $N-1$ bins and can be calculated as follows:

$$d_n = h_n - h_{n-1}, \text{ for } n = 2, 3, ..., N \tag{2.13}$$

The local minima of the histogram are located at distances, where the first-order derivative changes the sign from negative to positive (seen in Equation 2.14).

$$d_n \leq 0 \text{ and } d_{n+1} > 0 \tag{2.14}$$

Clustering

Clustering starts with multiple cluster centers and assigns every pixel to the nearest cluster center. The distance can be based on the color, texture, location and intensity of the pixels. After all pixels are assigned, the cluster centers are recomputed and the pixels are assigned to new cluster centers once again. This is done until the cluster centers are changed to less than a certain threshold. For 3D data, a good approach is to use the spatial distance of each pixel from the 3D cluster center as distance function. In [Wei14], a mean shift clustering is used to split the distance data into sub-regions.

Region Growing

Starting from multiple seed points, all close neighbors are added to the regions until there is no more left. Close can mean a similar distance or intensity value. For ToF data, the 3D spatial distance is used to determine if a neighboring pixel is close. A very crucial point for this method to succeed is the selection of the seed points. If frames are processed continuously, center points of regions in the previous frame are a good choice. For other regions the seed points can either be placed randomly or by using special algorithms. The authors in [Dal14] use a special region growing implementation for 3D data in their work. Region growing leads to good results, but also requires significantly more resources than simpler approaches such as the histogram segmentation.

Component Labeling

Component labeling is usually done with binary images. Connected components are merged, and the result is a picture with N non-connected components. This can be done using the 4-neighborhood or the 8-neighborhood. Component labeling is not directly applicable to 3D data, but a possible option is to divide the distance image into several slices of certain distance ranges, and create binary images for each slice. Component labeling is a low effort algorithm with the potential to satisfy the demands of certain simple applications.

2.3.3 Classification

A more advanced task is the classification of objects in images. In some applications it is important to recognize certain objects, for example the detection pedestrians, other cars or traffic signs. For more advanced use cases, already detected objects have to be re-recognized in continuous frames (tracking) in order to assign them trajectories and estimate their movement.

In other applications it is sufficient to detect whether there is an object within a certain area or not. With ToF sensor data it is easy to detect if there is a reflective object in front of the camera. The main goal of this thesis is object detection rather than object recognition. Thus, methods to classify and recognize objects are only briefly introduced here.

The first step to achieve object classification is usually to extract image feature points. These features are aimed to be as invariant as possible to translation, rotation, etc., in order to get robust feature descriptors such as Haar-like features, GiST features or Fourier features. Finally, classificators like SVMs or neural networks can then be applied to recognize certain objects. These classificators have to be trained with labeled training data prior to operation. For the classification step it can be beneficial to use the data of a 2D camera instead of the distance data, since the spatial resolution of ToF cameras is low in comparison [Nat08]. The fusion of the 3D and 2D images can improve the classification quality even further. Since a robust classification task takes high computational effort, it is currently not feasible for real-time operation on most embedded systems.

Chapter 3

Design

This chapter discusses the requirements of the system developed in this work, explains the used components in detail and presents a processing algorithm, planned to be used with the ToF system.

3.1 Requirements

The main goal of this work is to evaluate the feasibility of a ToF camera on a mobile platform. Since the performance of the sub-modules was not known in prior, the extent of the requirements strongly depends on the feasibility of the modules in the target environment. A non-specific breakdown of the basic requirements is listed here:

- **Platform:**
  The first requirement is to build up the platform and establish a communication between the scaled vehicle's control module and the ToF processing platform. The ToF platform shall have the ability to communicate with the scaled vehicle.
- **Distance data:**
  The distance shall be efficiently calculated on the automotive processing platform. The ToF camera itself provides raw data only, which is not directly usable for further processing.
- **Object detection:**
  When objects arise, the car control module should be informed as fast as possible. Thus, a fast algorithm to detect objects shall be implemented. The detection has to be robust in order to minimize false positive and false negative results.
- **Emergency braking:**
  If an object is detected and likely to cause a crash with the moving vehicle, the brakes of the car shall be applied to avoid the crash or at least reduce the impact of the collision.
- **Debug-ability:**
  The distance data obtained by the ToF camera shall also be possible to be viewed

on a computer in real-time. This makes it possible to analyze the obtained distance data in detail and move the algorithm development process to a PC.

3.2 Existing Platform

The prototype built in the work presented in [Dru15] was used as a starting point for this thesis. The contents of that work and the prototype are described in this section.

3.2.1 Time-of-Flight System

In the work published in [Dru15], the authors present a 3D ToF processing system for mixed critical systems consisting of an automotive microcontroller and a ToF camera. The AURIX (Automotive Realtime Integrated NeXt Generation Architecture) state-of-the-art microcontroller from Infineon Technologies provides three independent cores and supports safety-critical applications. The novel approach in this work is the implementation of the ToF data processing on this comparably weak platform, in regards to processing power and availability of high-speed memory.

Figure 3.1 shows the different processing blocks of this ToF application. The ToF camera illuminates the scene and captures the sensor data. This raw ToF data is then transferred to the automotive microcontroller where it is processed. One Central Processing Unit (CPU) core of the AURIX calculates the distance data from the raw sensor data. The other two cores are not used in the presented implementation, but reserved for future applications (e.g., pedestrian recognition). The implementation on the microcontroller is capable of processing the raw sensor data with frame rates up to 80 FPS.

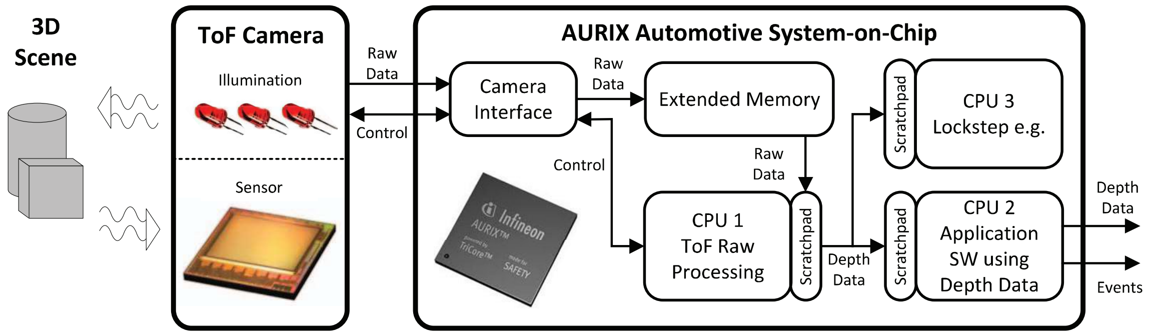


Figure 3.1: Functional blocks of the existing platform [Dru15].

Hardware

The prototype presented consists of a ToF camera evaluation board, the AURIX evaluation board and an adapter board to establish a connection between the two components. The adapter board connects the two evaluation kits via a special interface. A picture of the built-up platform is shown in Figure 3.2. The system is supplied by two Direct Current (DC) power adapters, one for each evaluation kit.

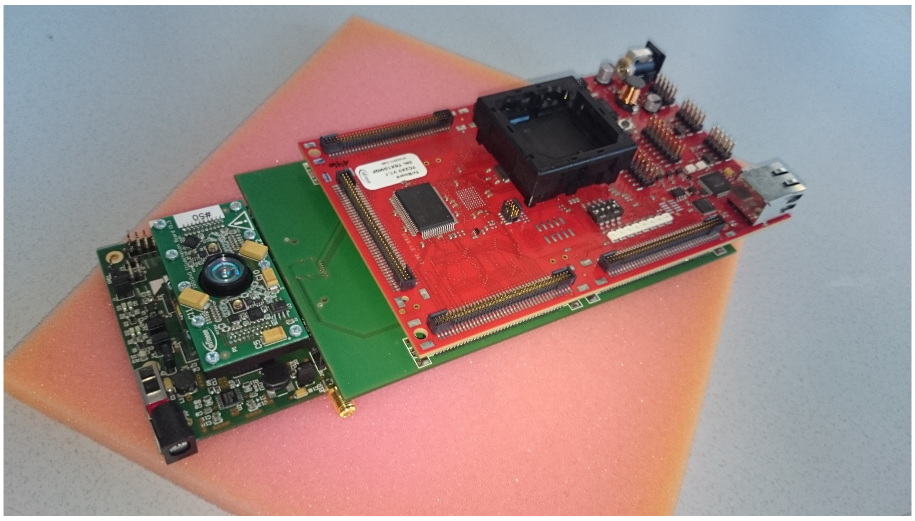


Figure 3.2: Photo of the existing platform. Obtained from [Dru15] with changes.

Software

The starting point of this work is a base implementation on the AURIX microcontroller, receiving raw sensor data (four phase-images) via the Camera Interface (CIF). The ToF camera is configured and started via the microcontroller. One core calculates the distance image (including global offset compensation) and a fast approximation of the amplitude image. Another CPU sends a stream containing the distance image and the amplitude image to a PC application for visualization.

3.2.2 Scaled Vehicle

The Virtual Vehicle research center in Graz has a Radio-Controlled (RC) 1/5 scaled vehicle available, which is used in this work. The platform is a modified version of a standard RC car, used as a simplified model of a real car. The module to control the engine and the actuators (brake servos, steering servos) are replaced by a custom control board. This board contains a microcontroller with a Controller Area Network (CAN) interface to communicate with other modules. A rear-view picture of the scaled vehicle can be seen in Figure 3.3.

3.3 Mobile Sensing Platform

This section describes the used hardware components in detail. The main components are the ToF image sensor evaluation kit, the AURIX evaluation board and the RC vehicle.

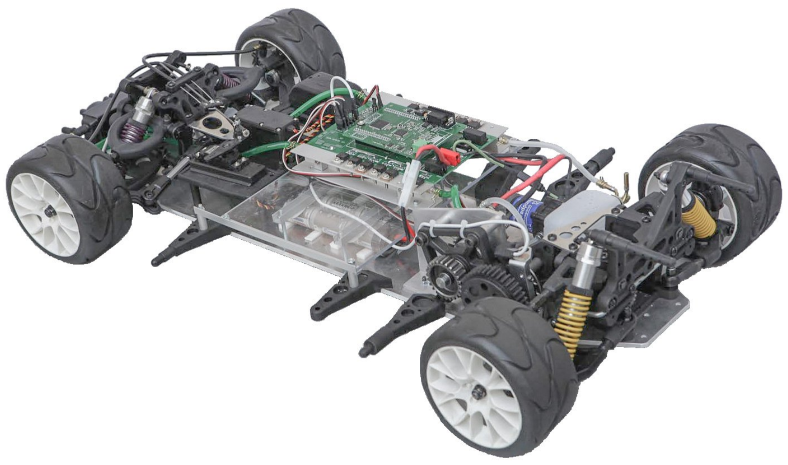


Figure 3.3: Scaled vehicle.

3.3.1 Overall System

Figure 3.4 illustrates the block diagram of the used modules and how they communicate with each other. The AURIX module configures and starts the ToF module and then continuously receives raw image data. The vehicle's control module sends the car state to the AURIX and can receive control signals to influence the current car state. Additionally, it should be possible to transmit internal processing data to a PC for visualization and debugging purposes.

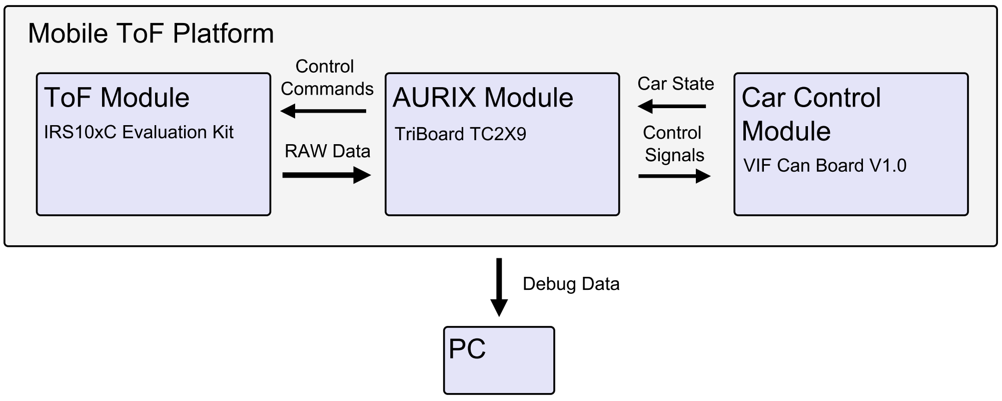


Figure 3.4: Block diagram of the main components.

3.3.2 ToF Module

As a platform to capture ToF raw data, the IRS10x0C evaluation kit from Infineon Technologies is used (see Figure 3.5). This evaluation kit comes with the IRS1020C, a 3D image sensor based on the ToF principle. The evaluation kit contains multiple distinct sub-modules mounted on the base-board. Those sub-modules are the image sensor board, and the illumination board. Those boards can be exchanged with other compatible boards using the same interface.

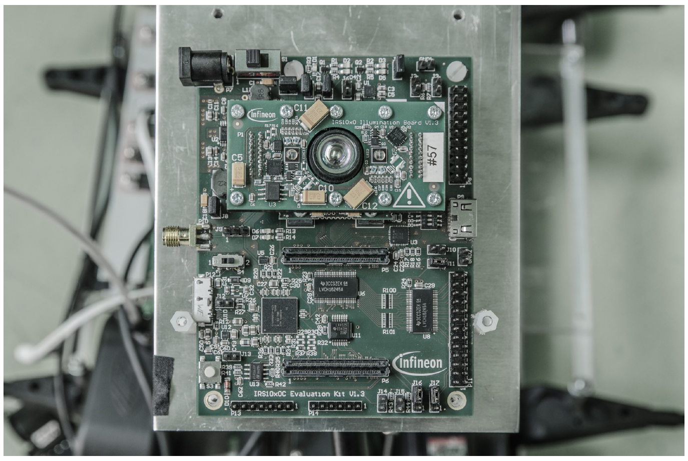


Figure 3.5: IRS10x0C evaluation kit.

The base-board offers a variety of debugging, testing and extension possibilities. It provides a USB interface, a Camera Serial Interface-2 (CSI-2) and multiple pin headers to transfer data to external platforms. The electrical current consumption of the system can be measured using a bridged pin-header. Many internal signals of the image sensor chip can be made available to the outside via one of the several debug interfaces. The General-Purpose Input/Output (GPIO) pins of the image sensor, mapped to debug pins on the evaluation kit can be configured to be assigned to various internal signals used during operation. There is also an interface for a Field-Programmable Gate Array (FPGA) on the board, which offers to directly plug in a Spartan-6 FPGA board to accomplish fast data processing. The kit has to be supplied by a 5 V DC source. While the illumination unit is active, the whole kit can temporarily consume electrical current peaks up to 2 A (depending on the use case). The used illumination board is equipped with two infrared LEDs with a centroid wavelength of 850 nm and a power consumption of $P_{tot} = 3.4\,\mathrm{W}$. With the combination of this illumination unit and the used image sensor, a minimum effective working range of 4 m can be expected.

3D Image Sensor

The IRS1020C, a 3D image sensor for consumer applications from Infineon Technologies AG and PMD Technologies AG is used in this work. This sensor uses the ToF principle to capture raw data, which can later be used to calculate the distance image. As already addressed in Section 2.2.4, the image chip offers a resolution of 352 x 288 pixels. But due to activated 2 x 2 binning, the mean of four pixel values are combined to one value directly on the imaging chip. The binning and the selection of a centered region of interest (ROI) decrease the spatial resolution to 160 x 120 pixels, but increase the SNR and the read-out speed.

Before the camera is started, the modulation frequency, the illumination time, the frame delay and the phase settings have to be set. In this work four phase measurements are performed with a phase setting of 0°, 90°, 180° and 270°.

Parallel Interface

The Parallel Interface (PIF) of the ToF image sensor consists of 12 data bits, a pixel clock (PIXCLK), a horizontal synchronization signal (HSYNC) and vertical synchronization signal (VSYNC). The PIF provides the data-stream of the 12 bit raw sensor phase-images.

The 26-pin PIF header and the FPGA connector of the ToF evaluation board contain all PIF signals. Thus, a connection via the FPGA connector or the PIF header can be used to receive the raw data from the sensor.

Additionally to the PIF, a CSI-2 interface is available on the ToF evaluation kit which can be used to receive data. It offers a faster read-out time and a lower vulnerability to electromagnetic interference at the cost of a higher complexity. But since the AURIX does not support CSI-2, the PIF is used to transfer sensor data.

Camera Configuration Interface

The ToF camera configuration and measurement setup is done via the Inter-Integrated Circuit (I^2C) interface of the sensor chip. The I^2C camera configuration interface is also included in both, the PIF header and the FPGA connector of the ToF evaluation kit.

Illumination Time

Since the system runs in an automotive environment, the illumination time has to be set low enough in order to avoid serious impact of effects like motion blur, motion artifacts and overexposure. But a low illumination time causes the SNR to decrease. To obtain the best performance, the illumination time shall be selected as high as possible, but low enough to not cause a significant impact on the negative effects of a high illumination time.

Different illumination times were evaluated in the targeted environment in order to determine a suitable value (see Section 5.2.1).

Modulation Frequency

The modulation frequency of the ToF camera determines the unambiguous range of the system. In this work, a time-consuming phase unwrapping algorithm is not implemented to save computation power for other tasks. Thus, the maximum distance is given by Equation 2.4 and wrong pixel values resulting from reflections beyond that distance shall be discarded. The Table 3.1 shows the unambiguous distance for different modulation frequencies in a relevant scope.

Lowering the modulation frequency increases the maximum distance of the system, but decreases the accuracy. With the given imaging sensor, the modulation frequency can be chosen within a range of 2.0833 MHz up to 100 MHz. The maximum unambiguous range has to be significantly higher than the working range. This avoids the wrong detection of objects further away than the unambiguous range and improves the detection of the phase-wrapped pixels.

Thus, different modulation frequencies were evaluated in order to find a good trade-off between accuracy and range for the given use case in this thesis (see Section 5.2.2).

Modulation Frequency (MHz)	Unambiguous Range (m)
30	5.0
20	7.5
17	8.8
12	12.5
10	15.0

Table 3.1: Unambiguous range for different modulation frequencies.

Lens

The ToF evaluation kit uses a fixed focus lens to focus the received light onto the image sensor. The field of view of the camera has to be considered for further examination. To calculate the 3D coordinates of a point, every camera pixel is assigned with a certain angle. For exact calculations, an angle for every pixel has to be determined. This can be done during calibration.

A less accurate approach to determine the angle of every pixel is to use an approximation. The horizontal field of view of the lens is 80°, while the vertical is 60°. For an image resolution of 160 x 120 pixels, a horizontal and vertical angle of 0.5° per pixel can be assumed. Although effects like distortion or aberration are not considered, this approach still results in workable data and can be implemented very efficiently.

3.3.3 AURIX Microcontroller

The TC299TF, a 32-bit microcontroller in the absolute high-end segment of the AURIX family from Infineon Technologies, is used in this work. This microcontroller uses three independent TriCore CPUs and is targeted to be used in automotive environments. The device operates at a frequency of 300 MHz and provides various amounts of different memories and built-in interfaces. Applications of the AURIX microcontroller are powertrain applications and safety applications in the automotive industry. Figure 3.6 shows the block diagram of the AURIX microcontroller.

TriCore Architecture

The TriCore architecture is based on the combination of a Reduced Instruction Set Computing (RISC) core, a microcontroller core and a Digital Signal Processing (DSP) core. The AURIX is the latest generation based on the TriCore architecture and offers three TriCore CPUs. The TriCore platform offers real-time capabilities and state-of-the-art safety features.

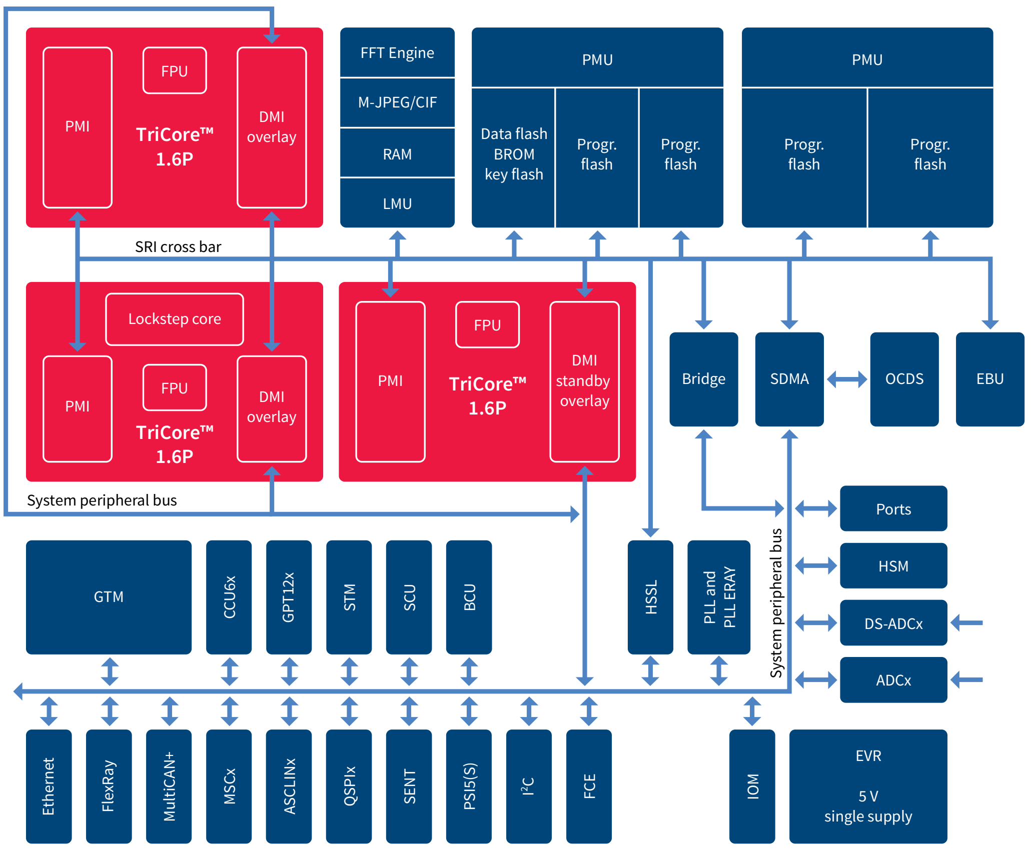


Figure 3.6: Block diagram of the AURIX [Inf16].

TriBoard TriCore Evaluation Board

The TriBoard TC2X9 provides access to many interfaces and functionalities of the AURIX microcontroller. Applications for the AURIX can be downloaded and programmed via the micro USB interface available on the board. To communicate with other systems, the board is for example, equipped with an RJ45 connector for Ethernet, and two CAN headers. Additionally, an on-board voltage regulator is used to generate the different supply voltages required for the microcontroller and the on-board peripherals. The evaluation board's supply voltage can vary from 5.5 V to 50 V. A picture of the evaluation board is shown in Figure 3.7.

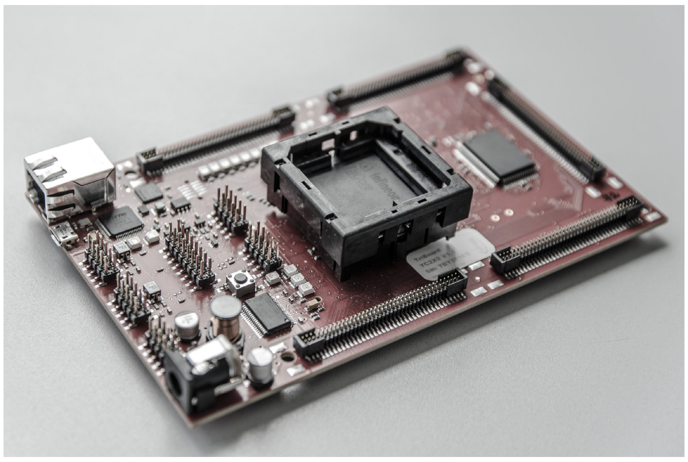


Figure 3.7: TriBoard TC2X9, TriCore evaluation board.

Memory

The available memory on the AURIX can be split up into data memory and program memory. Table 3.2 and Table 3.3 show the different available memories and their corresponding sizes.

- **Data memory:**
  The data memory is used for data storage during runtime. It can be further divided into the following different physical memories:
  - Data Scratch-Pad RAM (DSPR):
    Data read and write access to the local Data Scratch-Pad Random-Access Memory (RAM) of every CPU can be performed with an access latency of zero clock cycles. Thus, the DSPR is the best choice to achieve fast data access and should especially be used for frequently accessed data.
  - Local Memory Unit (LMU) RAM:
    There are 32 KByte of RAM available, provided by the LMU.
  - Extended memory (EMEM):
    Additionally, the EMEM can be used to save data. It is split up into the Application Data Memory (ADM) and the Extended Application Memory (XAM). The ADM is connected to the CIF module, which can directly write to the memory in hardware. For power saving reasons, the EMEM is in non-active mode after the power-on reset and has to be unlocked prior to use.
  - Data Flash (DFlash):
    The DFlash is also directly available on the chip, which can be used to permanently save data.
- **Program memory:**
  The program memory is used for program code storage and thus the instructions are fetched from it.

- Program Scratch-Pad RAM (PSPR):
  The PSPR can be used to store executable program code. Each CPU has a local PSPR, providing fast instruction fetching. Locating critical code in the local PSPR can significantly increase the performance of an application.
- Program Flash (PFlash):
  The PFlash is used per default to save the program code and constant data.

The DSPR and the PSPR of the TriCore CPUs can be addressed locally and globally. The local and global addresses are mapped to different regions in the memory map. Every CPU's scratch-pad memory is mapped to a different global address range and can be accessed separately. But the memories are also mapped to a local address range, which is the same for each CPU. Accesses to the local address range of the memory map are forwarded to the global address range of the active CPU. This is especially useful for the storage of data and code used on every CPU. Exclusive data and code should be stored only in the DSPR/PSPR of the respective CPU (global addressing).

Name	Size (KByte)
DSPR CPU 0	120
DSPR CPU 1	240
DSPR CPU 2	240
LMU RAM	32
EMEM	2048
DFlash	768

Table 3.2: Data memories.

Name	Size (KByte)
PSPR CPU 0	32
PSPR CPU 1	32
PSPR CPU 2	32
PFlash	8192

Table 3.3: Program memories.

Camera Interface

The AURIX provides a CIF, which is capable of transferring image data from a camera to the EMEM of the AURIX. The CIF implements 16 parallel data lines. A vertical synchronization signal (VSYNC), a horizontal synchronization signal (HSYNC) and a pixel clock (PIXCLK) are used to control the data flow. The CIF of the AURIX is used to connect the microcontroller with the ToF camera board, in order to receive sensor data.

Ethernet

The AURIX implements an Ethernet interface that is used with the RJ45 connector and a Gigabit Ethernet transceiver Integrated Circuit (IC) on the TriBoard. For Transmission Control Protocol/Internet Protocol (TCP/IP) functionality, the Lightweight IP Stack (lwIP), a small open source TCP/IP stack for embedded systems [Ada11] was used. The Ethernet module on the AURIX can be configured to run in 10 Mbit or 100 Mbit mode. The connection is used for debugging and visualization during the development process, and is not intended to be used in the final system. The lwIP stack can be used to initiate a network socket and send debug data to a PC.

Real-Time Operating System

The FreeRTOS [Rea11] is an open-source Real-Time Operating System (RTOS) for embedded systems and is used with the AURIX microcontroller. Since it is a very small and simple operating system, it offers only very basic functionality, and introduces very little overhead. The operating system provides real-time scheduling, inter-task communication and some synchronization functionality. A program can be written as a set of different independent tasks with given priorities. The scheduler ensures that the task with the highest priority always gets to work. If multiple tasks have the same priority, they share computation time in time slices. Synchronization between the tasks can be performed using semaphores and mutexes.

An official FreeRTOS port for the TC1782 microcontroller of the Infineon TriCore family exists and is freely available. In this work a slightly adapted version of this port is used with the AURIX TC299 microcontroller. The FreeRTOS scheduler runs concurrently on each of the three TriCores of the AURIX. The use of the FreeRTOS is very beneficial in the scope of this work since the provided functionality simplifies the developing process of real-time applications.

3.3.4 Scaled Vehicle

The Virtual Vehicle Research Center modified a 1/5 scaled gasoline-powered vehicle to be used with an electrical engine. The RC car can be driven using a remote control. The maximum speed v_{max} of the scaled vehicle is about $20\,\frac{\text{km}}{\text{h}}$. Additionally, the car has a microcontroller on board that is in charge of controlling the car's engine and actuators.

Power Supply

The scaled vehicle has two power supplies. A 7.2 V, 2300 mAh Nickel–Metal Hydride (NiMH) battery for the electrical engine and another 7.2 V, 2000 mAh NiMH battery for the electronic circuits. The separation into two independent supply circuits is required, because the electronic components need a very stable supply voltage. The electronic circuit containing the engine cannot provide a very stable voltage level, since the voltage drops in high load situations of the engine.

Scaled Vehicle Microcontroller

The scaled vehicle comes with a low cost, low power Atmel 8-bit AVR RISC based microcontroller with an integrated CAN controller [Atm08]. The controller reads the output signals from the RF receiver module, which are originally directly sent to the steering/braking servos of the car and to the motor-control block. The microcontroller manipulates the signals and forwards them to the actuators. Depending on the current mode, motor control signals are generated and sent to the custom designed engine control board.

The microcontroller permanently monitors significant values of the vehicle; for example the battery voltage, the motor voltage or the rotation speed. Additionally, it measures the temperatures at multiple points on the scaled vehicle. Figure 3.8 shows a picture of the vehicle's microcontroller mounted on top of the engine control board.

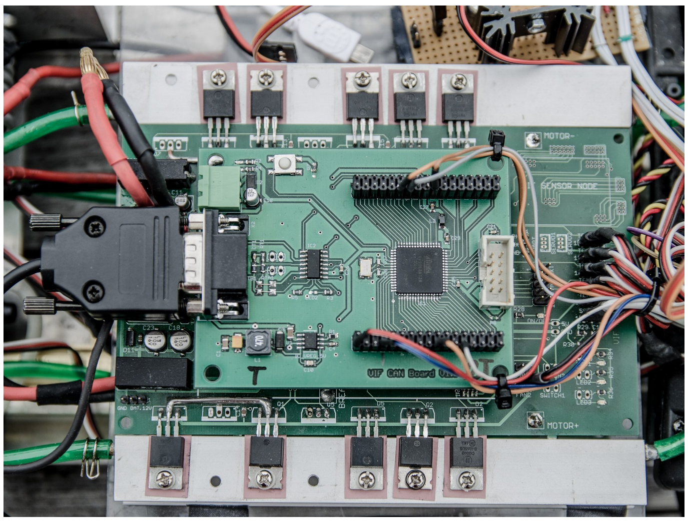


Figure 3.8: Vehicle microcontroller.

Remote Control

The scaled vehicle can be controlled with a radio frequency (RF) remote, working at a frequency of 40 MHz. The remote has three different channels. One for accelerate/brake, one for steering left/right and one for forward/backward mode. Figure 3.9 shows a picture of the remote control.

Speed Sensor

The scaled vehicle is equipped with a speed sensor. It is a photoelectric sensor that detects the holes in a metal plate, assembled to the engine shaft. The rate of detecting holes is

Figure 3.9: RF remote for the scaled vehicle.

proportional to the rotation speed of the wheels, and thus also to the current speed of the car. This assumption is of course only true as long as the tires do not slip. Since the holes in the sensor are evenly distributed, it is not possible to detect whether the car goes forward or backward. Figure 3.10 shows a picture of the speed sensor.

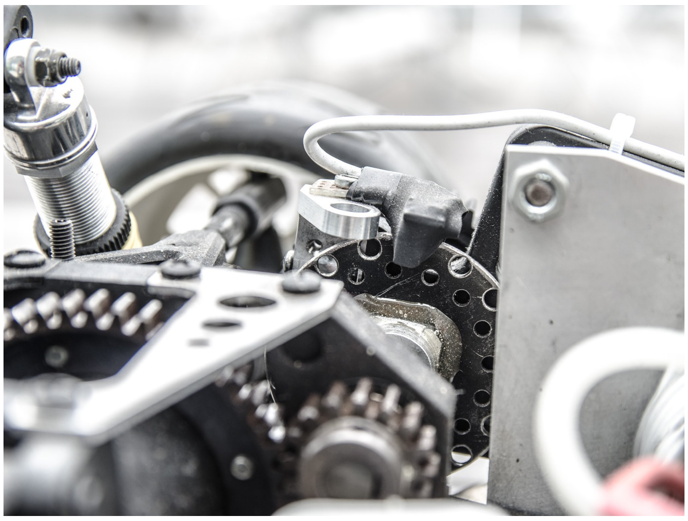


Figure 3.10: Speed sensor of the scaled vehicle.

Camera Mount

The camera is mounted on the front of the scaled vehicle pointing in forward direction, since the system's aim is to be used for pre-crash detection. Other than that, the position shall be comparable to the mount on a fully sized car, what would most likely be in the windshield or somewhere else in the front.

CAN Interface

The scaled vehicle's microcontroller implements a CAN node and is able to send and receive CAN messages. The interface is used to send the current car state to other bus participants. The car state includes a number of values available on the vehicle's microcontroller. For example the motor current, the remote control values, the battery voltage, different temperatures and the rotation speed. It is also possible to control the vehicle via the CAN bus instead of the remote control.

The automotive microcontroller of the ToF processing system is connected to the vehicle's microcontroller via a CAN bus. The purpose of this connection is that the ToF processing system is able to receive the car's current state and to send control commands.

3.4 Emergency Braking

This section first explains the components of the stopping distance, and then focuses on the processing time requirements for the ToF processing system in order to implement emergency braking.

3.4.1 Stopping Distance

The total stopping distance s_{stop} of a car is the sum of the reaction distance s_{react}, and the braking distance s_{brake}. This holds true for a human driven car, as well as for an autonomously driven car.

$$s_{stop} = s_{react} + s_{brake} \tag{3.1}$$

Reaction Distance

The reaction distance is the distance, the car moves from the time an obstacle appears until the brake is applied. For a human driver, this reaction time depends on a variety of factors, like fitness, age, alertness and fatigue. In regular circumstances, the reaction time of a human is less than 1.5 s. The reaction distance s_{react} depends on the currently driven speed v and the reaction time T_{react}. The calculation of the reaction distance is shown in Equation 3.2.

$$s_{react} = v \cdot T_{react} \tag{3.2}$$

Braking Distance

The braking distance s of a car can be calculated using the current velocity v, the friction coefficient μ and the gravitational acceleration $g = 9.81\,\frac{\mathrm{m}}{\mathrm{s}^2}$ as shown in Equation 3.3.

$$s = \frac{v^2}{2\mu g} \tag{3.3}$$

The Table 3.4 shows the braking distance for different speeds of a car when using a typical friction coefficient for a car tire and dry road ($\mu = 0.7$). As seen in the table, the braking distance is proportional to the square of the speed. When braking from a high speed ($> 75\,\frac{\text{km}}{\text{h}}$), the braking distance will most probably be the dominating component of the stopping distance. This non-linear relationship has to be kept in mind when transitioning the results from the scaled vehicle to a fully sized vehicle.

Velocity ($\frac{\text{km}}{\text{h}}$)	Velocity ($\frac{\text{m}}{\text{s}}$)	Braking distance (m)
10	2.8	0.56
20	5.6	2.2
30	8.3	5.1
50	13.9	14.0

Table 3.4: Braking distance for different speeds on a dry road.

3.4.2 Automated Braking

According to Section 3.3.4, the maximum speed v_{max} of the scaled vehicle is assumed to be $20\,\frac{\text{km}}{\text{h}}$. The maximum braking distance $s_{brake,max}$ for the scaled vehicle on a dry road ($\mu = 0.7$) can be calculated using Equation 3.3 and results to:

$$s_{brake,max} = \frac{v_{max}^2}{2\mu g} = 2.25\,\text{m} \tag{3.4}$$

It can be assumed that all objects are robustly detected within a distance of 4 m from the lens (according to Section 3.3.2). The scaled vehicle should be able to stop before an obstacle on a dry road, this means the range $s_{stop,range}$ for stopping the car is:

$$s_{stop,range} = 4\,\text{m} \tag{3.5}$$

For the ToF processing platform, the reaction time T_{react} depends on a sum of times and its composition is stated in Equation 3.6. The object capturing time $T_{obj,capt}$ is the time it takes from a newly appearing object until it is captured and available for processing. T_{proc} is the time to process an image and decide whether to brake or not. $T_{brake,delay}$ is the time from sending the brake signal until the brake gets activated physically.

$$T_{react} = T_{obj,capt} + T_{proc} + T_{brake,delay} \tag{3.6}$$

This time is only valid, if the system performs the image processing and decision-making for every image independently. If multiple sequential frames are required to draw a decision, the reaction time increases.

With the use of Equation 3.3, 3.5 and 3.6, the maximum reaction time $T_{react,max}$ can be calculated as stated in Equation 3.7. $T_{react,max}$ describes the maximum time delay for the used platform to activate the brake after an object appears, in order to avoid a crash.

$$T_{react,max} = \frac{s_{stop,range} - s_{brake,max}}{v_{max}} = 0.315\,\mathrm{s} \tag{3.7}$$

This means the ToF processing system has to activate the brake within a time span of 0.315 s in order to successfully avoid a collision.

Required Frame Rate

In the worst case, an obstacle occurs directly after the ToF sensor data (four phase-images) is captured. Thus, it takes the whole frame delay time T_{FPS} until the frame containing the obstacle is even captured. Afterwards, the three CPUs sequentially perform their processing part until the signal to brake the car can finally be sent.

Assuming ideal pipelining, the processing time on each of the three CPUs $T_{proc,CPU}$ is equal the frame delay time T_{FPS}. The time from sending the brake signal until the brake gets physically activated $T_{brake,delay}$ can be roughly estimated with 100 ms.

Considering all this assumptions, an estimation of the worst case reaction time $T_{react,wc}$ can be calculated with:

$$\begin{aligned} T_{react,wc} &= T_{FPS} + 3 \cdot T_{proc,CPU} + T_{brake,delay} \\ &= 4 \cdot T_{FPS} + 0.1\,\mathrm{s} \end{aligned} \tag{3.8}$$

The worst case reaction time $T_{react,wc}$ has to be smaller than the maximum allowed reaction time $T_{react,max}$ from Equation 3.7:

$$T_{react,wc} \leq T_{react,max} \tag{3.9}$$

With the use of Equation 3.8 and Equation 3.9 and $T_{react,max}$ from Equation 3.7, the frame delay time T_{FPS} has to meet the following criteria:

$$T_{FPS} \leq \frac{T_{react,max} - 0.1\,\mathrm{s}}{4} = 53.8\,\mathrm{ms} \tag{3.10}$$

If the system should still provide its functionality in the case that the object detection fails once in a while, another FPS delay T_{FPS} has to be added, resulting in the following criteria:

$$T_{FPS} \leq \frac{T_{react,max} - 0.1\,\mathrm{s}}{5} = 43\,\mathrm{ms}$$

To meet the requirements, the system's frame delay time has to be lower than 43 ms, corresponding in a minimum frame rate of about 23 FPS. To allow even more margin for error, a frame rate of at least 25 FPS should be used with the system. Thus, the maximum processing time on each CPU must not exceed 40 ms in order to meet the deadlines.

3.5 ToF Processing Algorithm

This section describes the processing algorithm to detect and react to possible obstacles in front of the car. Figure 3.11 shows the flow chart of the algorithm, with the input data and output data of each stage. Each step of the algorithm is described in detail in this section.

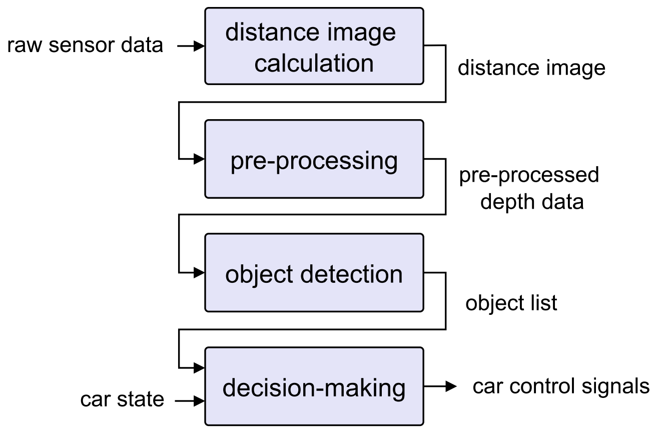


Figure 3.11: Data flow of the processing algorithm.

3.5.1 Distance Image Calculation

The first step of the algorithm is to calculate the distance image from the raw image data of the sensor. The amplitude image is not entirely calculated to save resources, since it is only used for thresholding. Figure 3.12 shows the data flow of the distance image calculation. The single steps performed during this task are described in more detail.

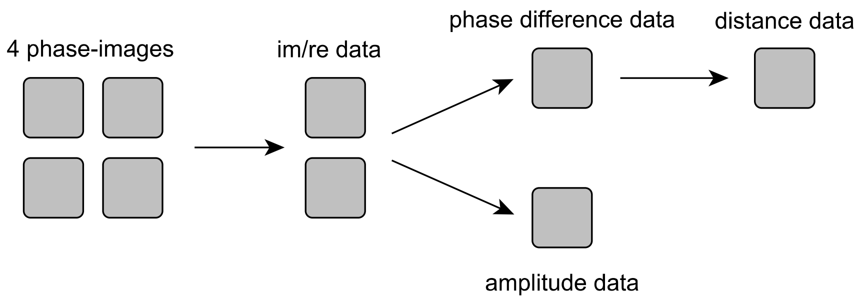


Figure 3.12: Distance image calculation.

Distance Image

After the four phase-images from the sensor are available, the arctangent function has to be calculated for each pixel in order to determine the phase difference $\Delta\varphi$ (see Equation 2.2). The distance can be calculated using Equation 2.3. Since the only variable in that equation is $\Delta\varphi$, the phase to distance conversion is a simple multiplication with the constant *RangeFactor*.

$$d(x, y) = \Delta\varphi \cdot \mathit{RangeFactor} \tag{3.11}$$

The *RangeFactor* can be calculated with the speed of light c_0 and the modulation frequency f_{mod}:

$$\mathit{RangeFactor} = \frac{1}{2} \cdot \frac{c_0}{2\pi \cdot f_{mod}} \tag{3.12}$$

Global Offset Compensation

The global offset is an additive constant for all pixels, and is determined during calibration. It can be distributed all over the unambiguous range and depends on the modulation frequency. Thus, the compensation of this offset is a mandatory step within the distance image calculation.

To achieve global offset compensation, the global offset d_{global} has to be subtracted from every distance value $d_{in}(x, y)$ of the image. The distance values after the global offset compensation $d_{go}(x, y)$ are calculated with:

$$d_{go}(x, y) = d_{in}(x, y) - d_{global} \tag{3.13}$$

After the global offset compensation there might be values outside the unambiguous range D_{unamb}. Thus, an unambiguous range shift has to be applied to obtain the correct distance values $d_{out}(x, y)$ within the unambiguous range.

$$d_{out}(x, y) = d_{go}(x, y) - \left\lfloor \frac{d_{go}(x, y)}{D_{unamb}} \right\rfloor \cdot D_{unamb} \tag{3.14}$$

Figure 3.13 shows the distance values before the global offset compensation, after the global offset compensation and after the unambiguous range shift. In the illustration an unambiguous range of 5 m ($f_{mod} = 30\,\text{MHz}$) and a global offset of $d_{global} = -1\,\text{m}$ is given.

For example: a real distance value of 0.5 m (in green segment) will result in a distance value of 4.5 m after the uncompensated distance image calculation. The global offset compensation adds 1 m to every distance value. Accordingly, the distance after the compensation is 5.5 m. This value is outside the unambiguous range. In the next step, the unambiguous range shift is performed to map all values back into that range. After the unambiguous range shift, the distance value is 0.5 m, equal to the real distance.

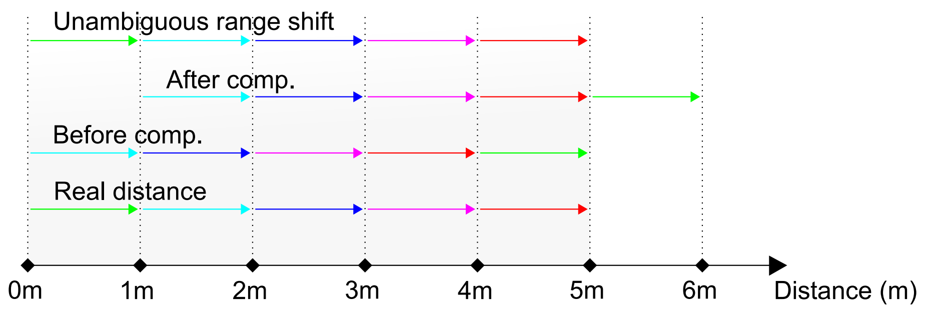


Figure 3.13: Global offset compensation and unambiguous range shift.

3.5.2 Pre-Processing

Several pre-processing steps are performed to improve the quality of the distance image. During this procedure the image is smoothed, the noise is reduced and erroneous pixels are discarded in order to improve the expressiveness of the distance image for further processing. Invalid pixels are marked with a certain out-of-range value to signalize the further processing steps to not consider those pixels. The data flow of the pre-processing steps can be seen in Figure 3.14. All steps performed during pre-processing are introduced here.

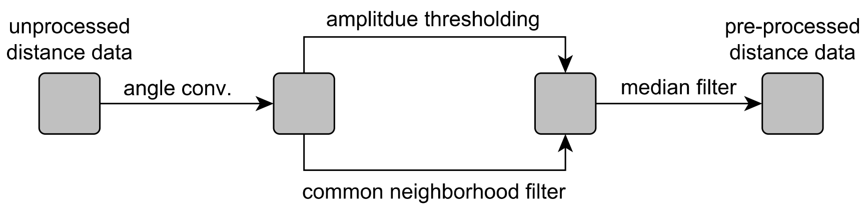

unprocessed distance data
angle conv.
amplitdue thresholding
common neighborhood filter
median filter
pre-processed distance data

Figure 3.14: Pre-processing.

Camera Angle Conversion

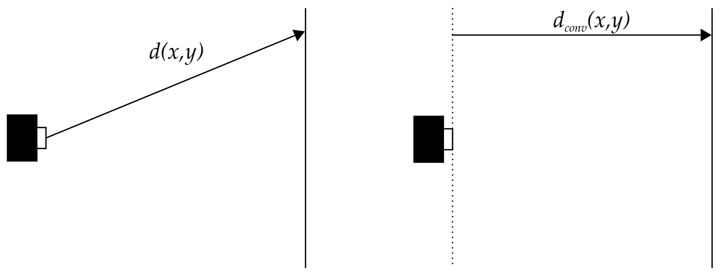
Up to this point, the distance image contains the distance values from the reflected object to the camera sensor for every pixel. This causes points of a plain surface parallel to the image sensor to result in different distances. To obtain the distance values from a plain instead of a point, the distance values have to be converted. Figure 3.15 shows the difference of the distance data before and after the conversion. The camera angle conversion is performed because it improves the performance of the object detection.

An approximation for the converted distance value $d_{conv}(x, y)$ can be calculated with the use of the horizontal pixel angle β and the vertical pixel angle α, as shown in Equation 3.15. The Figure 3.16 shows the camera angle conversion and the included horizontal and vertical angles.

$$d_{conv}(x, y) = d(x, y) \cdot cos(\alpha) \cdot cos(\beta) \tag{3.15}$$

The calculation of the vertical pixel angle α and horizontal pixel angle β is stated in Equation 3.16. The coordinates $(x,\ y)$ define the horizontal and vertical pixel position of the 160 x 120 pixel image. As already mentioned in Section 3.3.2, an approximated angle of 0.5° is assumed for every pixel.

$$\begin{aligned}\alpha &= (x - 60) \cdot 0.5^\circ \\ \beta &= (y - 80) \cdot 0.5^\circ\end{aligned} \tag{3.16}$$



(a) Before. (b) After.

Figure 3.15: Camera angle conversion.

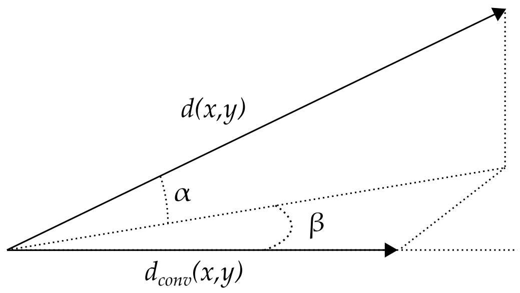


Figure 3.16: Pixel angle.

Amplitude Thresholding 1

The amplitude value of a certain pixel indicates the quality of the measurement. The first pre-processing step is to discard pixels with low amplitude since they are very likely to be erroneous. An easy method in achieving this has been introduced in Equation 2.10.

Since the amplitude is not uniquely distributed over the whole image, a reference image is used as a base for the thresholding. Figure 3.17 shows the amplitude reference image of a white surface at a distance of 1 m.

The idea is to discard pixels with amplitude values $A(x, y)$ smaller than the amplitude reference image, scaled with a constant scaling factor c_{th1}. Pixels meeting the following criteria are discarded:

$$A(x, y) < A_{ref} \cdot c_{th1} \tag{3.17}$$

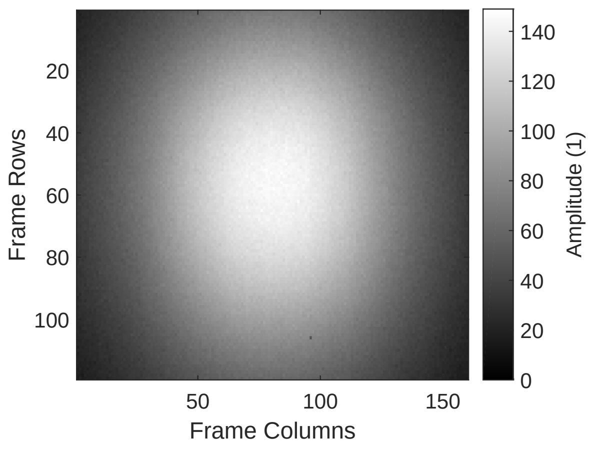


Figure 3.17: Amplitude reference image.

Amplitude Thresholding 2

A second thresholding method is applied in order to avoid phase-wrapping. This method compares the threshold value with the amplitude value of the pixel, weighted with its corresponding squared distance. This thresholding method takes into account that objects at a higher distance cause lower amplitudes in the detecting pixel. Since the amplitude decreases with the square of the distance (see Equation 2.5), it is weighted with its squared distance value in order to compensate this effect. If an object outside the unambiguous range is detected, phase-wrapping occurs and the detected distance value is too small. Thus, pixels where phase-wrapping occurred are likely to be identified and discarded in this step.

The amplitude values $A(x, y)$ are weighted with the squared distance $d(x, y)^2$ and compared with the amplitude reference image, scaled with a constant scaling factor c_{th2}. Pixels meeting the following criteria are discarded:

$$A(x, y) \cdot d(x, y)^2 < A_{ref} \cdot c_{th2} \tag{3.18}$$

Common Neighborhood Thresholding

CN thresholding is applied to further reduce the noise of the distance image. This method takes the output of the camera angle conversion as input, since it has to be applied to an image with all pixels present.

After the application of CN thresholding and amplitude thresholding, both outputs are combined. A pixel marked as invalid in one of the two output images is marked as invalid in the combined output as well.

Median Filter

Afterwards a median filter is applied to the distance values, to obtain a smoother version of the image. The median filter requires sorting the neighborhood values of the considered input pixel.

3.5.3 Object Detection

At this step all remaining pixels of the pre-processed distance image are assumed to be trustworthy, so the actual object detection can be performed. Multiple steps are executed to segment the image and detect whether there are obstacles in front of the ToF camera or not. Figure 3.18 shows the data flow of the object detection.

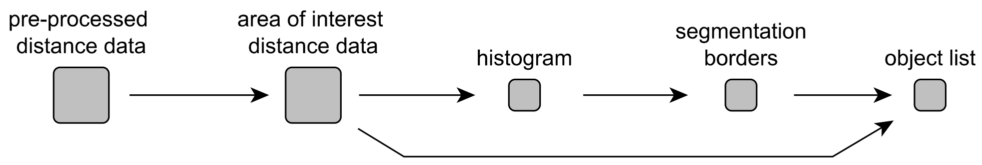


Figure 3.18: Object detection.

Area of Interest

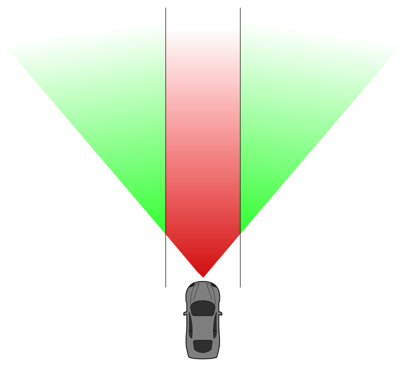
Not every pixel in the field of view of the camera has to be considered for object detection. Objects too far away, or not in the direct impact area of the car can be discarded. Examples for areas outside the AOI are the ground plane, bridges, walls or trees next to the road.

First the range of interest is considered. Distance values further away than a certain maximum distance are likely to be erroneous and are discarded. Also pixels with a very low distance value (up to a few centimeters) are very likely to be noise and are discarded as well.

$$d(x,y) = \begin{cases} d(x,y) & \text{, if } d_{min} < d(x,y) < d_{max}, \\ \text{invalid} & \text{, otherwise.} \end{cases} \tag{3.19}$$

Afterwards the perspective is considered. The vehicle should only be stopped if there is an object in front of it. So pixels outside the impact area of the car are discarded and not considered in future steps. Figure 3.19 shows a vehicle's AOI, if only the objects in the impact area of a straight driving car are to be considered. The pixels from objects in the green zones are discarded.

Since every pixel of the camera is assigned with an angle, it can be calculated if a pixel is within the AOI or not. The horizontal field of view (80°) and the vertical field of view (60°) have to be considered.


(a) Top view.

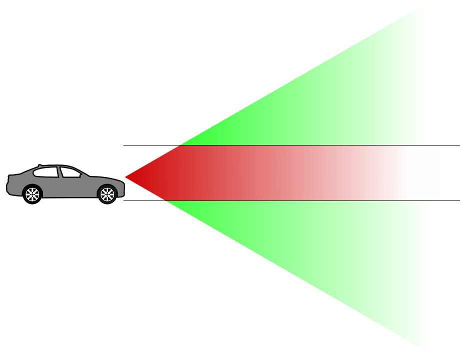

(b) Side view.

Figure 3.19: AOI of the scaled vehicle.

Equation 3.20 shows the mathematical statements to determine if a detected distance value is within the AOI or not. The horizontal distance $D_{horizontal}$ and the vertical distance $D_{vertical}$ are constants, based on the width and height of the vehicle. The Figure 3.20 depicts the relationship between the distance of a reflected object from the camera plane $d_{conv}(x, y)$ (after the camera angle conversion) and the horizontal or vertical deviation from the center caused by the angle (α and β) of the corresponding pixel.

$$
\begin{aligned}
d(x,y) &= \begin{cases} d(x,y) & \text{, if } d(x,y)\cdot\tan(\alpha) \leq \frac{D_{vertical}}{2}, \\ \text{invalid} & \text{, otherwise.} \end{cases} \\
d(x,y) &= \begin{cases} d(x,y) & \text{, if } d(x,y)\cdot\tan(\beta) \leq \frac{D_{horizontal}}{2}, \\ \text{invalid} & \text{, otherwise.} \end{cases}
\end{aligned}
\tag{3.20}
$$

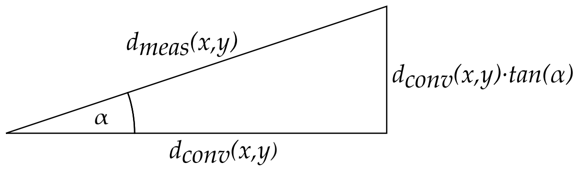


Figure 3.20: AOI calculation.

Histogram Segmentation

At this point, a histogram segmentation based on the approach presented in Section 2.3.2 is performed. First, a distance histogram is created with the remaining pixels in the AOI. The histogram is smoothened using averaging over multiple bins. Afterwards, the first derivative of the histogram is determined, by subtracting two consecutive histogram values. The zero-crossing of the first derivative is used to obtain the local minima of the histogram, in order to get the segmentation of objects located at different distances.

All local minima of the histogram are evaluated for their feasibility to form a segmentation border. If the histogram values within two segmentation borders meet special criteria, the corresponding pixels of those histogram intervals are extracted to an object.

This approach leads to under segmentation. If two objects are located at the same distance, they are most likely recognized as one. Nevertheless, the histogram segmentation is used because it is very efficient.

Each object obtained in the histogram segmentation is assigned an x, y and z coordinate and an area. The coordinates are the mean values of all points of the object. The area is determined by the number of pixels assigned to the object. Additionally, the minimum and maximum coordinates of the objects are determined in order to identify the objects limits. Output of the histogram segmentation is an object list containing the mentioned properties, describing the form and the location of every object.

3.5.4 Decision-Making

The final step in the algorithm is to decide if it is required to take over the control of the scaled vehicle. The decision-making algorithm has two inputs, the car state and the object list. The state of the car is periodically sent to the ToF processing system and includes the rotation speed and remote control values. The object list is available each time the processing of a new ToF image is completed.

When new input data is available, the decision-making algorithm is triggered and evaluates the situation. If all of the following requirements are met, control take-over is initiated:

- At least one of the detected objects exceeds a specified area, depending on the distance.
- The braking distance at the current speed exceeds a certain fraction of the distance to the closest relevant object in the impact area.
- The car's driving direction is forward and the speed is higher than a certain minimum speed.

The algorithm itself is built in the structure of a state machine (see Figure 3.21). In the "User Control" state, it is evaluated whether one of the detected objects is likely to cause a collision if the vehicle continues driving at the current speed. If there is a dangerous situation detected, the state machine translates to the "Stop via CAN" state and the vehicle's brakes are applied. At the time the car stands still, the state translates back to the "User Control" state and can be controlled via the remote again.

If no new data (either image data or a new car state) has been received within a certain period of time, the system is supposed to be in a non-functional state. Then the state translates to the "System Not Working" state and the vehicle is stopped by sending a brake command via a CAN message.

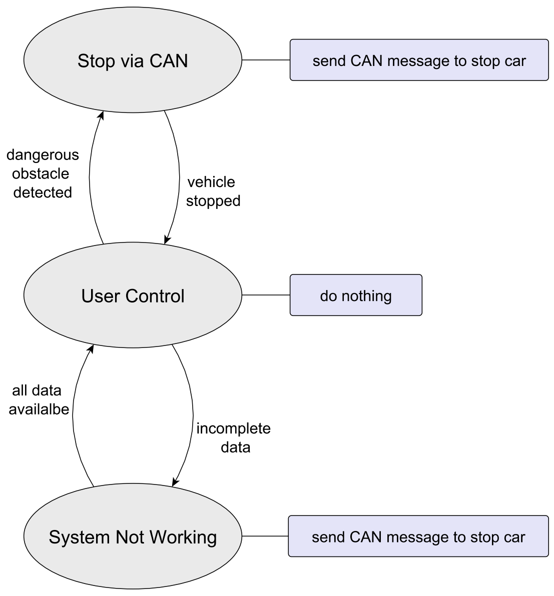


Figure 3.21: State machine for decision-making.

3.6 Debug System

One important requirement of the system is debug-ability. With the ability to access the internal data of the system, it is possible to rate the quality of the sensor data and the implemented algorithm. Additionally, it is possible to visualize processed data.

Included Data

The following data shall be available for debugging:

- **Debug data:**
  Every CPU can write debug data to a debug string available on every CPU.
- **Current car state:**
  The current state of the car is transferred to the AURIX via CAN. This data is included into the debug string of each CPU.
- **Histogram:**
  All or a part of the distance image's histogram. This data is included into the debug string of each CPU.
- **Metadata:**
  The metadata includes some settings of the camera and some other constants. This is useful to interpret saved data for later use.
- **Phase-images:**
  The four raw phase-images from the sensor are transferred. The mode is used to obtain the raw sensor data externally. But since the amount of transferred data is high, it cannot be used at high frame rates.

- **Real and imaginary image:**
  In that mode the real and imaginary image are calculated from the four phase-images. This compresses the transferred images to two, without losing information and makes it applicable for higher frame rates.
- **Distance image:**
  Only the distance image is sent.
- **No data:**
  In this mode, no data is sent, to remove the overhead and to achieve a higher frame rate. This comes with the loss of observability of the system.

Since the transfer of that data introduces a significant processing overhead, the structure of the debug data should be able to be customized. At some stages of the development process it might be convenient to have most of the debug data available, at the cost of a higher processing time. At other phases a fast system with as little delay as possible will be required.

PC Workstation

With the received debug data on a workstation, a number of applications are possible:

- **Debugging:**
  An obvious application is to use the permanently transferred data for debugging. During the processing, each CPU can write specific intermediate data to a debug string, which is sent to the workstation along with every frame. Images after certain processing steps can be transferred to the workstation and visually verified.
- **Visualization:**
  Another application is to use the workstation as platform to display the distance data (e.g., while the car is driving). If the car moves around, a mobile device has to be used that can be placed on the car.
- **Save sequences:**
  While the car moves, a mobile workstation can record the received data sequences from the system. With the obtained data (depth image, car state, etc.) it is possible to analyze the performance and debug the algorithm “offline”.
- **Emulation:**
  The exact same algorithm running on the embedded system is implemented on the PC workstation. Since the raw sensor data can be received, it is possible to move the development process of the algorithm to the workstation. Changes of the algorithm can be performed and tested in a short space of time compared to deploying and testing on the embedded system.

Data Rate

Depending on the amount of debug data transferred, the data transmission can become a very time consuming task and even limit the maximum operating speed of the whole system. Here the required data transfer rates to transfer four phase-images in time are explained in detail. One 16 bit image with a size of 160 x 120 pixels, results in 38400 byte of data. Four 16 bit phase-images contain 153600 byte of data.

The data rate R of four 16 bit phase-images of data N_{data} sent at a frame rate f_{rate} is calculated with:

$$R = N_{data} \cdot f_{rate} \tag{3.21}$$

Frame Rate (FPS)	Data Rate ($\frac{\text{Mbit}}{\text{s}}$)
10	12.29
20	24.58
30	36.86
40	49.15

Table 3.5: Data rate for different frame rates.

The Table 3.5 shows the minimum data rate required to not exceed the deadlines given by certain frame rates. In practice, the transmitted data will also include overhead in form of headers and checksums and thus the minimum frame rate has to be even higher than the values stated here. Additionally, the processor might be occupied with other tasks and thus cannot use the full frame delay time for transmission. The data rate of the Ethernet interface of the AURIX is limited to a certain value, depending on various factors (like lwIP configuration or the used memory). This limitation has to be kept in mind, when choosing the debug data and the frame rate.

Chapter 4

Implementation

This chapter is about the implementation details of this thesis. First the development process and the used tools are presented. The major part of the chapter is split into the detailed description of the hardware segment and the software segment of this thesis. The chapter ends with a description of the debug system.

4.1 Development

In this section, the development tools that are mainly used are described. Afterwards the workflow to implement the requirements and to improve the functionality is presented.

4.1.1 Tools

During the development process, multiple software tools were used. The most important ones are introduced here. Additional programs are mentioned in Appendix A.6.

Free TriCore Entry Tool Chain

The Free TriCore Entry Tool Chain from HighTec [Hig15] is an Eclipse based integrated development environment used in this work. The main part of the toolchain is the TriCore Development Platform based on the GNU development tools. It can be used to create executables for TriCore microcontrollers and comes with a C/C++ cross compiler and a debugger with multi-core support. Additionally, it is possible to program the DFlash and the PFlash within the development environment. For this thesis, the Free TriCore Entry Tool Chain is used to write, organize and debug the code for the AURIX microcontroller.

Matlab

Matlab [Mat15] is used to visualize the distance data and debug the processing algorithm implemented on the AURIX. The software has many required functions already built-in (e.g., plotting images, creating histogram). Thus, it is easy to write a debug application within a short time. Using scripts, it is possible to receive User Datagram Protocol (UDP) data via Ethernet and to calculate and display the distance image. But this easy to use, high-level structure of Matlab can also cause problems. The execution performance of Matlab scripts is slow compared to similar implementations in other programming languages. The available routine to receive data via UDP can not handle a livestream of the distance data in real-time. In addition, it is not possible to perform image processing calculations in real-time.

Nevertheless, Matlab is used to receive and store single images of the livestream, inspect the quality of the data, perform image processing steps, and quickly display images and debug data.

Visual Studio

Caused by the shortcomings of Matlab, Visual Studio [Mic12] was used to develop a C# Windows application with the purpose of supporting the algorithm development on the AURIX microcontroller. Due to the higher performance of the C# application, the distance data can be received in real-time via UDP. Additionally, the data is able to be processed and visualized without violating any deadlines. Thus, the C# application is the main tool used to debug and visualize data from the embedded system. The disadvantage compared to Matlab, is the significantly higher effort to create and change an application since almost all functionality has to be implemented manually.

The application is also used to receive a livestream of the raw ToF phase-images and move the whole algorithm development process to the PC. The advantage is that debugging on PC is far easier and the whole compilation/execution process is faster. The algorithm in C# is written in very similar to C code, in order to easily move the code to the AURIX. A more detailed explanation of the C# application developed with Visual Studio can be found in Appendix A.7.

4.1.2 Workflow

The following list presents the different working steps that were performed during this thesis.

- **Base implementation:**
  The first step was to understand and use the base implementation of the existing platform described in Section 3.2. The existing implementation was created using the Tasking toolchain from Altium. Since the Free TriCore Entry Tool Chain is used in this thesis, the existing code also had to be adapted.

- **Implement debug interfaces:**
  Next step was to implement the ability to observe the system's state. The base version was extended to provide debug functionality via Ethernet using the UDP protocol. With the use of UDP debugging, it is possible to validate the single image processing steps done on the embedded system.

  The CAN functionality was activated in order to send and handle received CAN Messages via the CAN interface on the TriBoard. With the use of a CAN to USB converter, CAN messages can also be used to send short debug messages and view them on a PC. The used application to send and receive CAN messages from a PC is introduced in Appendix A.6.

  A third way to gain information about the current state of the processing algorithm on the embedded system is to use the GPIO pins. Special port pins can be set high during certain processing steps. This also enables the possibility to obtain detailed timing information of the processing algorithm on the microcontroller. A special USB logic analyzer is used to make the signal traces directly available on a PC. The tool is further described in Appendix A.6.

- **Camera mount and configuration:**
  Different positions of the whole ToF imaging platform on the scaled vehicle were evaluated in order to obtain the best possible distance image quality. Additionally, appropriate camera parameters (illumination time, modulation frequency) were picked for the targeted use case.

- **Image processing algorithm:**
  The development of the image processing algorithm was an iterative process starting with the raw sensor data. The following steps were run through multiple times:

  - Development:
    Every modification of the algorithm was first implemented, validated and debugged in high-level language (Matlab scripts or C#) on the PC workstation. This was done using the raw sensor data as input, obtained via a UDP debug stream. Special parameters of the algorithm were tweaked and evaluated in real-time using a Graphical User Interface (GUI). This process is illustrated in Figure 4.1.

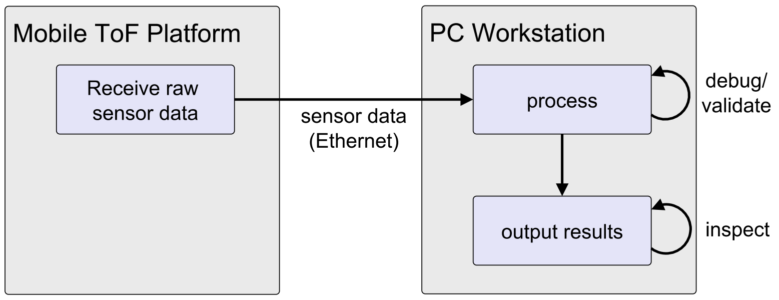


Figure 4.1: Algorithm development on workstation.

- Validation:
  After major changes, the algorithm was deployed to the embedded system. The raw sensor data and output data of the image processing algorithm were sent to the workstation in order to validate the result. The output data can for example include a processed distance image, a histogram or the object-list. Additionally, the timing of the different tasks was measured using a logic analyzer. The timing data was then examined to meet the processing deadlines. This is especially important, since there is no timing information available from the emulation on the workstation. The validation process of the algorithm running on the AURIX is shown in Figure 4.2.

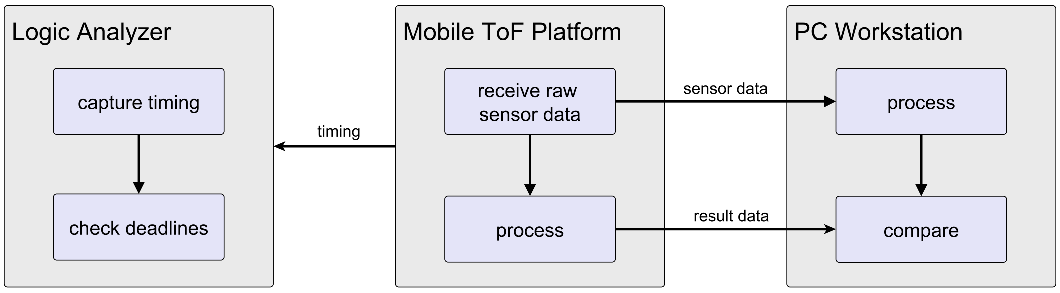


Figure 4.2: Algorithm validation.

- **Decision-making algorithm:**
  The decision-making algorithm was mainly evaluated directly on the running system, with the real-world car state available via CAN messages. But a basic evaluation of the algorithm was also possible with simulated CAN messages using the CAN-to-USB converter.

- **Test run:**
  The whole system with the most recent algorithm was tested in real scenarios, in order to evaluate the functionality/performance of the system.

The whole development process was an iterative process, starting from a very simple base version. For major modifications of the algorithm, the influence to the performance of the whole system was evaluated. Shortcomings discovered within the validation steps were analyzed and aimed to be eliminated, by applying modifications in prior developing steps.

4.2 Hardware Platform

This section describes the detailed composition of the built-up platform. The mounting of the ToF camera on the scaled vehicle is described and the adapter board used to connect the ToF camera and the AURIX evaluation board is presented.

4.2.1 Overall System

The overall system consists of the scaled vehicle with the ToF processing platform mounted on it. The scaled vehicle comes with on-board control modules. Two battery packs and a voltage converter are available on the scaled vehicle in order to supply the single components with power. A block diagram of the overall system is depicted in Figure 4.3. The figure shows the different building blocks of the scaled vehicle with the ToF processing platform.

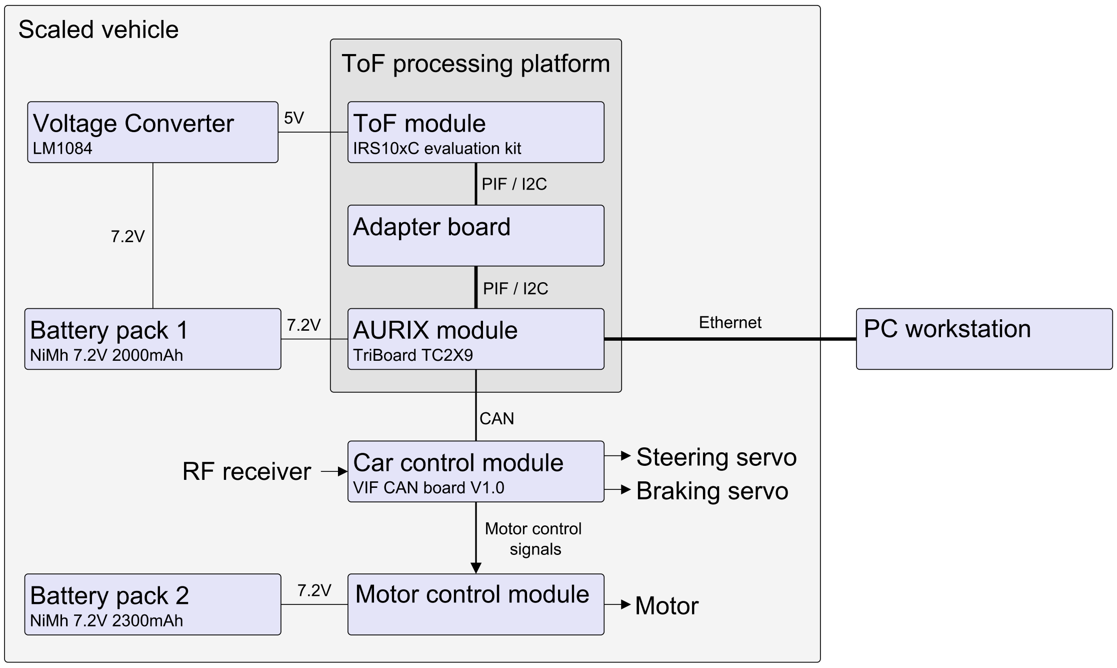


Figure 4.3: Block diagram of the overall system.

4.2.2 Power Supply

The battery pack for the electronic circuits consists of six 1.2 V NiMH cells. Thus, the nominal voltage is 7.2 V. In practice, the voltage decreases from about 7.5 V to 6.0 V during the discharging process. The TriBoard with the AURIX has a voltage regulator on-board and can be directly supplied with the 7.2 V from the battery. Since the ToF evaluation kit requires a supply voltage of exactly 5.0 V, there is a need of an external voltage regulator for the camera evaluation board. A small board with a linear low-dropout voltage regulator (LM1084, Texas Instruments) is used to convert the battery voltage to 5.0 V with an output current up to 5 A.

The voltage converter board does not work very efficiently as a lot of energy is wasted through the linear conversion. But a fast and simple solution is sufficient for the prototype built in this work. A more sophisticated approach, using an efficient step-down converter to extend the battery runtime is presented in Appendix A.2.

4.2.3 Adapter Board

The ToF evaluation kit and the AURIX TriBoard are connected using a special adapter board. The first version (V 1.0) of the adapter board uses two high speed hermaphroditic terminal/socket strip connectors to directly connect the corresponding pins of the AURIX TriBoard with the FPGA header of the ToF evaluation board. The FPGA header includes the signals of the PIF and the camera configuration interface of the ToF evaluation kit.

An alternative approach is to establish a flexible connection between the AURIX TriBoard and the ToF evaluation kit using a ribbon cable. This connection simplifies the positioning of the ToF camera on the scaled vehicle. A second version of the adapter board connects the pins of the PIF header of the ToF evaluation kit to the corresponding pins on the AURIX. The 26-pin PIF header includes the signals of the PIF and the camera configuration interface of the ToF evaluation kit.

In both approaches, the signals of the PIF on the ToF evaluation kit are connected to the CIF pins of the AURIX. Since the CIF provides 16 data bits and the PIF only provides 12 bits, the data lines are connected Most Significant Bit (MSB)-aligned. The I^2C camera configuration interface pins from the ToF evaluation kit are connected to one of the I^2C interfaces of the AURIX. The signal assignment is illustrated in Figure 4.4. Both connecting methods offer the possibility to configure the ToF camera and receive raw sensor data.

The flexible connection, using the second version of the adapter board, leads to a significantly decreased image quality due to its vulnerability to interference. The issues occurring with this setup are described in Appendix A.1. Thus, the first version is used in this thesis, approving the less flexible, and bulkier composition.

4.2.4 Camera Mount

The ToF evaluation kit and the AURIX TriBoard are coupled using an adapter board. The whole ToF processing platform is rather bulky and difficult to place on the small scaled vehicle. Hence a flexible camera arm is used to bring the ToF camera system into position. This construction also allows evaluating different positions of the camera on the car. A picture of the camera mount is shown in Figure 4.5. The ToF processing platform is attached to an aluminum plate fixed to the camera arm. With that setup, it is possible to evaluate different positions of the camera, while it is still possible to heavily tighten the camera arm to minimize shaking while driving.

4.3 Interfaces

The overall system consists of multiple submodules which have to communicate with each other. This section describes the different interfaces used to connect the components.

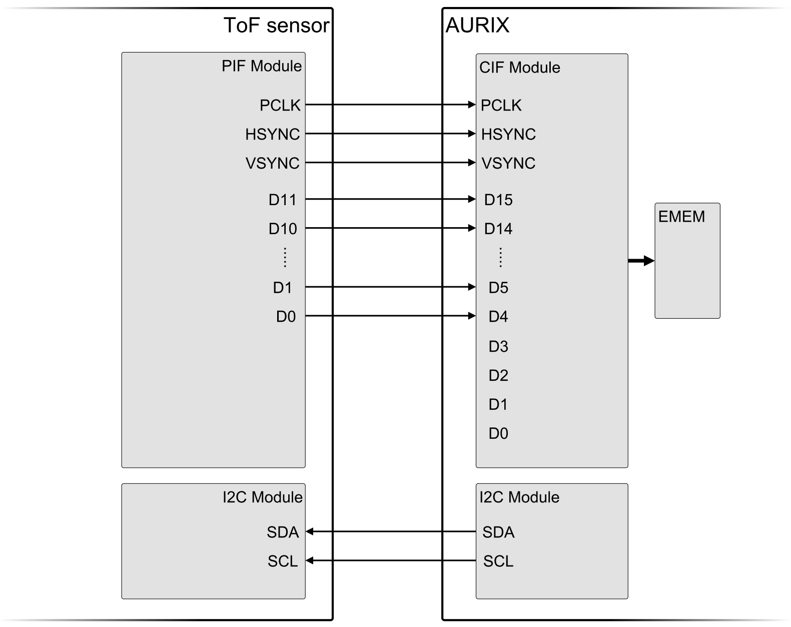


Figure 4.4: Signal assignments to establish a communication between the AURIX and the ToF image sensor.

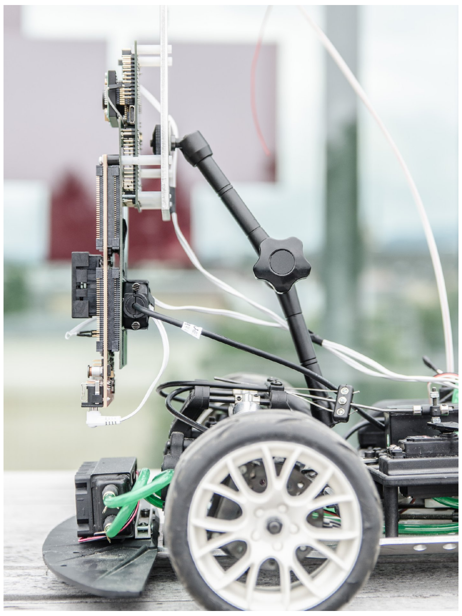


Figure 4.5: Flexible arm on the scaled vehicle.

4.3.1 CAN Interface

A CAN connection is established between the AURIX microcontroller and the control board of the scaled vehicle (see Figure 4.3). The microcontroller periodically sends multiple CAN messages about the current car state and the current values from the remote control channels.

There is also the possibility to take over control of the car via the CAN bus. If the CAN take-over message is received, the car microcontroller changes in “CAN Control Mode”. The take-over message is a regular CAN message with a special ID and a data field containing the car control signals. In “CAN Control Mode” the car control values from the CAN take-over message are considered, instead of the values from the remote channels. The CAN control mode ends when a special CAN give-back message is received or a timeout occurs.

The two operating modes of the scaled vehicle’s microcontroller are illustrated in Figure 4.6. The detailed structure of the CAN messages is explained in Appendix A.4.

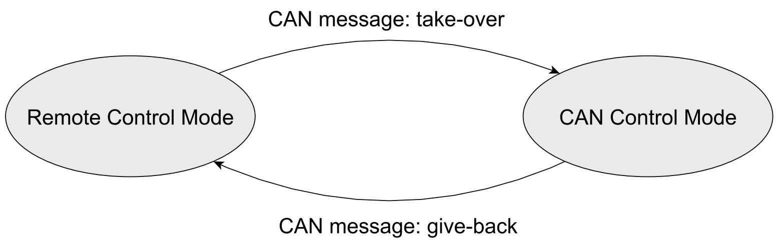


Figure 4.6: Different operating modes of the scaled vehicle.

4.3.2 Camera Control Interface

The I^{2}C camera configuration interface is used to configure the ToF data before the image capturing is started.

An array of camera configuration data is stored on the AURIX and transferred to the ToF camera when initiated by the AURIX. The array containing all used register values to configure the ToF sensor is listed in Appendix A.10.

In this work, the main parameters of interest within the camera configuration are the modulation frequency, the illumination time and the frame rate. Since those parameters are adjusted frequently, the configuration of the required registers is explained in detail.

Modulation Frequency

The modulation frequency can be adjusted by setting multiple register values containing the phase-locked loop (PLL) settings. There exist four different PLL presets that can be configured for different modulation frequencies. The aimed PLL preset has to be selected by defining the register value in the sequence settings for the respective phase-image.

The register parameters shown in Listing 4.1 are used to set the modulation frequency to 17 MHz. The PLL preset 3 is set to 17 MHz and selected in the sequence settings of the four captured phase-images.

```
{CFGCNT_S00_PLLSET, 0x0002},
{CFGCNT_S01_PLLSET, 0x0002},
{CFGCNT_S02_PLLSET, 0x0002},
{CFGCNT_S03_PLLSET, 0x0002},
{CFGCNT_PLLCFG1_LUT3, 0x6449}, // 17 MHz
{CFGCNT_PLLCFG2_LUT3, 0x2762}, // 17 MHz
{CFGCNT_PLLCFG3_LUT3, 0x03F6}, // 17 MHz
```

Listing 4.1: Settings for a modulation frequency $f_{mod} = 17\,\text{MHz}$.

```
{CFGCNT_S00_EXPOTIME, 0x109A}, // 2ms @ 17MHz
{CFGCNT_S01_EXPOTIME, 0x109A}, // 2ms @ 17MHz
{CFGCNT_S02_EXPOTIME, 0x109A}, // 2ms @ 17MHz
{CFGCNT_S03_EXPOTIME, 0x109A}, // 2ms @ 17MHz
```

Listing 4.2: Settings for an illumination time $T_{illu} = 2\,\text{ms}$.

```
{CFGCNT_S00_FRAMERATE, 0x0000},
{CFGCNT_S01_FRAMERATE, 0x0000},
{CFGCNT_S02_FRAMERATE, 0x0000},
{CFGCNT_S03_FRAMERATE, 0x1000}, // ca 30 FPS
```

Listing 4.3: Settings for a frame rate $f_{rate} = 30\,\text{FPS}$.

Illumination Time

The illumination time can be defined for every phase-image within the sequence separately. The actual illumination time resulting from that register value depends on the active PLL preset.

The parameters shown in Listing 4.2 are used to set the illumination time to 2 ms. These values are only valid at a modulation frequency of 17 MHz.

Frame Rate

The frame rate can be set via an adjustable delay after a phase-image is captured. The ToF camera is configured to capture a sequence of four phase-images in a loop. Thus, the frame delay after the first three phase-images is set to zero. And with the value of the frame delay after the fourth phase-image, it is possible to define the frame rate.

The required parameters to define a frame rate of about 30 FPS are shown in Listing 4.3.

4.3.3 Parallel Sensor Interface

The ToF camera is configured to continuously capture four consecutive phase-images and send them via the PIF to the AURIX. The CIF of the AURIX receives the images in hardware and directly writes them into the ADM EMEM. After the four frames are received and written to the ADM, an interrupt is raised to notify the AURIX that new sensor data is available.

Figure 4.7 shows the timing of the PIF for an illumination time of 1 ms and a pixel clock of 66.6 MHz. In the illustration an active task is indicated by a "high" signal. As seen in the figure, an interrupt is only raised after all four phase-images were fully transferred.

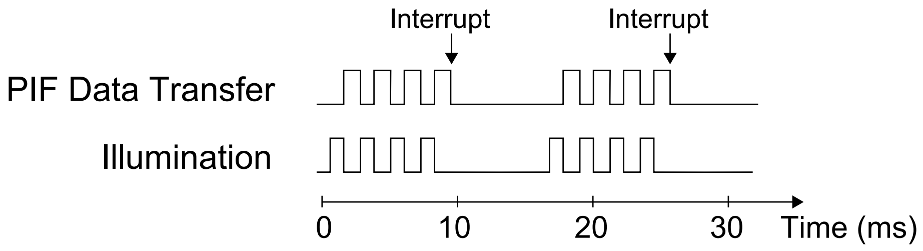


Figure 4.7: Timing of the PIF.

The illumination time can be changed by modifying the camera configuration via the I^2C serial bus. For a pixel clock of 66.6 MHz, the time to read out one 16 bit phase-image and transfer it via the PIF, equals to 1.23 ms. This time is also the minimum time between two illuminations. In dynamic environments the time between two illuminations should be kept as low as possible in order to avoid effects like motion artifacts.

4.3.4 Debug Interface

Here the debug connection between AURIX and the PC workstation is explained in detail. The AURIX microcontroller board and a PC workstation can be connected using an Ethernet connection. Data is sent from the AURIX to the PC to obtain observability of the state of the embedded system. Due to its simplicity UDP was used to transmit the data. In this section the structure of the UDP packages is described.

UDP Transmission

After the processing algorithm is finished, CPU 0 is ready to send relevant and related data of the processed image. The included data can be selected via pre-processor defines in the AURIX code. There are different modes of included data-sets predefined on the AURIX. To change the mode, the AURIX code has to be re-compiled and programmed to the AURIX flash.

Depending on the included debug data, different processing steps have to be performed. For certain modes it is possible to suppress the further algorithm calculation in order to achieve higher frame rates. For example, there exists a processing mode that only sends the four phase-images via UDP and does not process any data. Using that mode makes it possible to record raw data at an acceptable frame rate.

UDP Data Structure

Debug data is sent from the AURIX to the PC as a sequence of UDP packages. In order to keep the complexity low, all sent UDP packages have a size of exactly 400 bytes. Since various data is sent via UDP, the data flow has to be marked with control packages in order to determine which data is currently being received.

The structure of the data always follows the same pattern:

- **Start frame:**
  To signal the receiver that a new data sequence is transmitted, a special start package is sent. The start package has to start with the string “`FRM_STRT`” and the remaining bytes in order to fill up the 400 byte package do not matter.
- **Debug data:**
  Directly after the start frame, debug data can be sent. The debug data typically consists of four UDP packages. The first package contains metadata of the AURIX system. The other three packages contain debug data of each CPU. For example, the debug data of CPU 0 includes information about the current car state, since that data is available on that CPU.
- **Image data:**
  To inform the receiver that image data will be transmitted next, a control UDP package starting with “`IMG_x`” has to be sent. Afterwards the actual image-data is sent, which might consist of many UDP packages. There can be multiple images transferred. The “x” in the image control package has to be incremented for every image, starting with 0.
- **Stop frame:**
  After the last image was completely transmitted, a special UDP package starting with the string “`FRM_STOP`” has to be transferred in order to inform the receiver that the data sequence is over.

Figure 4.8 shows the structure of a valid sequence of UDP packages. In that case, two images are transferred. As indicated in the illustration, the debug data, image 0 and image 1 usually consist of multiple UDP packages while the control packages only consist of one package each.

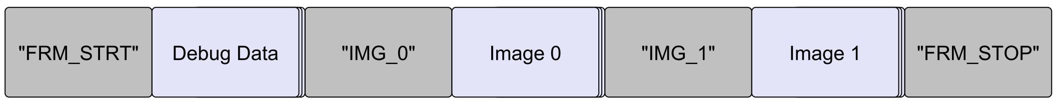


Figure 4.8: UDP transmission structure.

UDP Reception

For reception of the debug data, the received UDP frames have to be reassembled to gain the original data structure. Figure 4.9 shows a state diagram for a receiver to save the debug data.

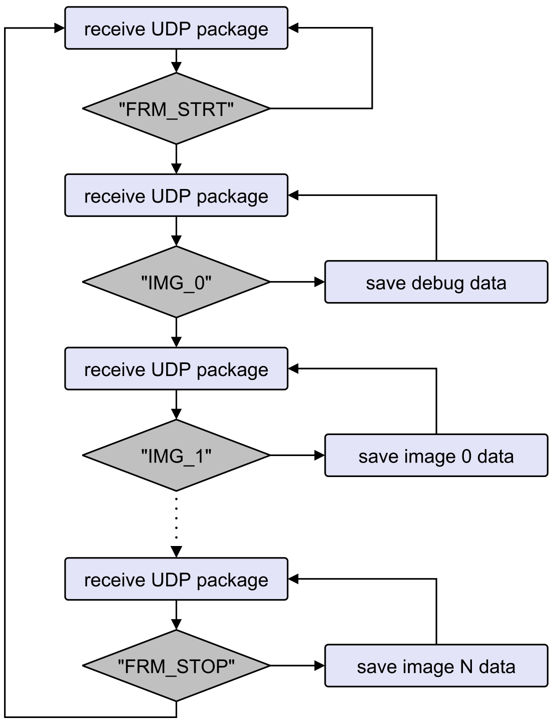


Figure 4.9: UDP receive state-machine.

4.4 AURIX Implementation

This section is about AURIX specific implementation details. It discusses the partitioning of the algorithm tasks onto the three CPUs and the distribution of the data within the memory. Additionally, the buffering process for the UDP transmission is presented.

4.4.1 CPU Partitioning

Figure 4.10 shows the partitioning of the main processing tasks on the three AURIX CPUs.

After four raw phase-images are transferred from the ToF Camera to the EMEM, CPU 1 gets notified (via an interrupt) and starts to process the images. CPU 1 calculates the distance data and pre-processes the image. Then the output image is saved to a shared memory and an interrupt in CPU 2 is triggered. CPU 2 fetches the output image and continues to process the image in order to obtain an object list. The object list is saved to a shared memory and an interrupt in CPU 0 is triggered. CPU 0 starts processing either if a new object list or a new car state is available. Either way, the decision-making routine is performed in order to decide whether to brake or not. Every time a new ToF image was fully processed, CPU 0 sends debug data via Ethernet.

A good way to increase the performance of the system is to use pipelining. The total workload has to be partitioned as equally as possible among the CPUs in order to obtain

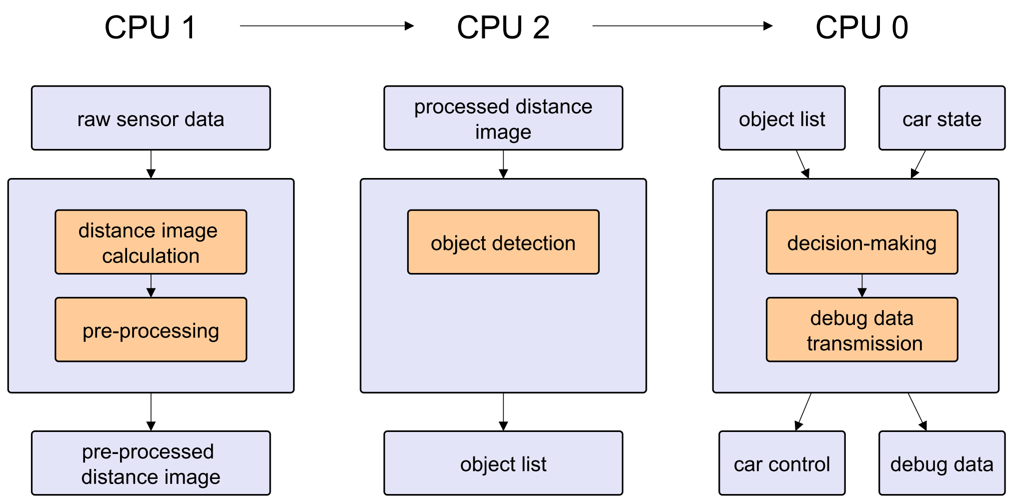


Figure 4.10: CPU partitioning.

a high level of parallelism and utilization. This causes an increased throughput of the system and increases the maximum frame rate. Figure 4.11 shows the pipelining concept. At the time CPU 1 has finished its processing part and CPU 2 takes over, CPU 1 is already available to process the next sensor data from the ToF camera.

The ideal case would be to divide the workload in a way that each CPU takes exactly the same processing time T_{proc}. In that case a maximum frame rate of $f_{rate} = \frac{1}{T_{proc}}$ can be achieved without violating deadlines.

4.4.2 Shared Memory

To transfer data between two CPUs the following different approaches are possible. Two methods are described here:

- Image data is transferred using the XAM EMEM as intermediate storage. Smaller data amounts are transferred via the LMU RAM (object list, etc.).
- Another method is to transfer data using the DSPRs of the different CPUs. This method requires additional synchronization to avoid data inconsistency and thus is not implemented in the final system.

To transfer data using the above mentioned method, the sending CPU has to copy the data from the DSPR into the temporary memory. Then the sending CPU triggers a software interrupt in the receiving CPU, causing a task to unblock. This task then copies the data from the temporary memory into its DSPR.

There is still the chance that the receiving CPU has not finished reading when the sending CPU already writes new data again. Thus, a synchronization flag is introduced, to avoid writing into currently used memory areas. The flag is set by the writing CPU right before writing new data to the region. The flag is reset by the receiving CPU, after copying is done.

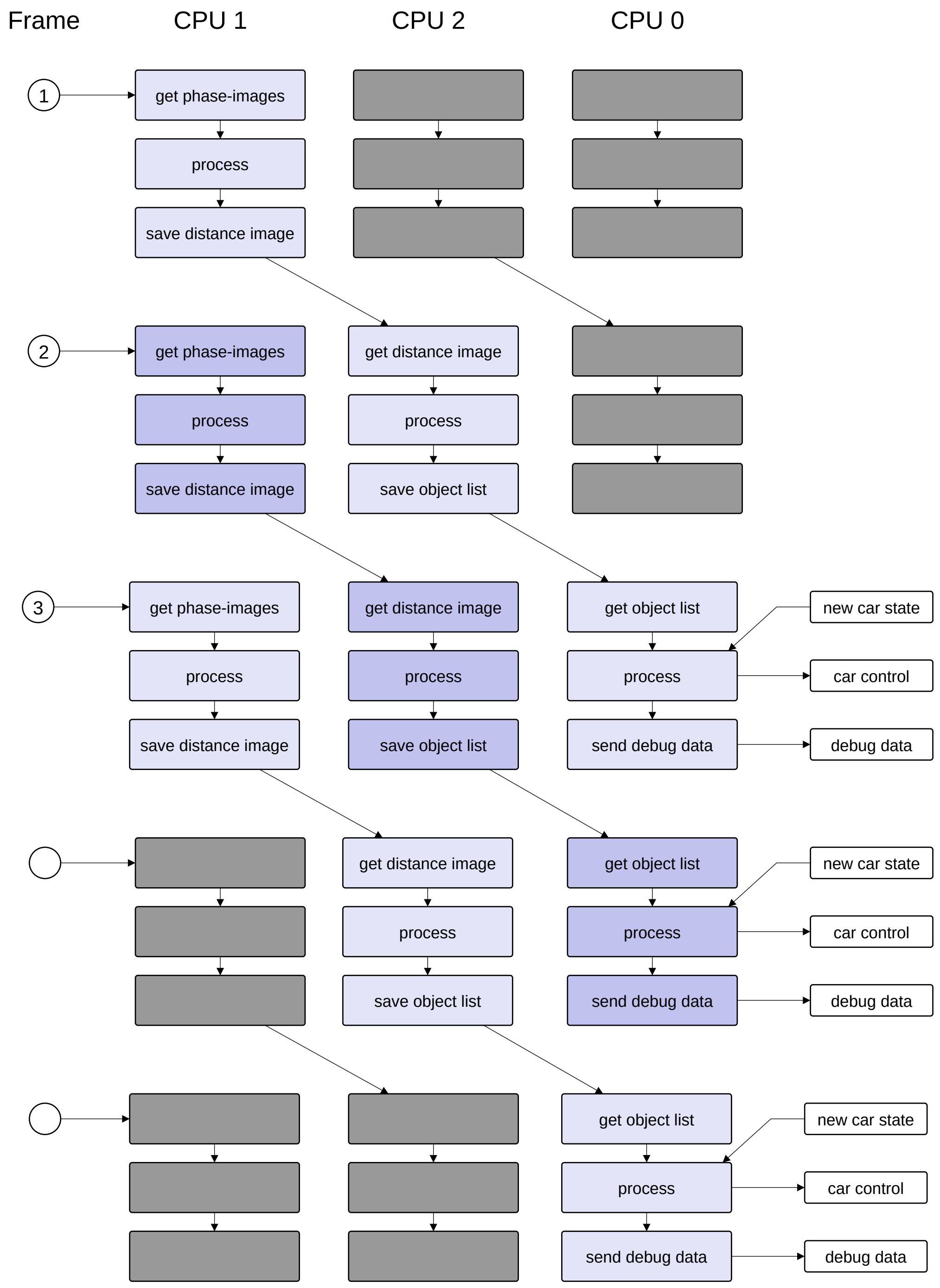


Figure 4.11: CPU pipelining concept.

If the frame rate is set to a reasonable value the processing algorithm will have no problems to meet the deadlines. Since the accesses to the shared memory is temporally well-organized, the synchronization flag is not essential during regular operation.

4.4.3 Memory Partitioning

The memory configuration of the AURIX can be controlled via the linker description file. Figure 4.12 gives a simplified overview of the memory partitioning on the AURIX. Since the image data occupies the majority of memory space, the reserved space for image data in the different memories is emphasized in the illustration.

The following memories are used to store program code and data:

- **DSPR:**
  The memory used by the RTOS is mapped equally into each CPU's DSPR using its local address range. This is beneficial since the RTOS code is used by all three CPUs. The remaining part of the DSPR is used independently and thus addressed globally. This causes only variables that are actually used in the code of one CPU to occupy space in its memory.

  The image slots on CPU 1 and CPU 2 are used for image processing, for example to save a temporary image after a filter operation. The image slot on CPU 0 is used as high speed buffer to send image data via UDP. The remaining DSPR memory is used to save other data (like the car state, debug data, etc.) or is unused.

  A solution to separate the memory placement onto the different CPUs is shown in Appendix A.9.
- **LMU RAM:**
  The LMU RAM is used to share data between the CPUs, for example synchronization variables.
- **EMEM:**
  The raw phase-images from the ToF camera, received via the CIF of the AURIX are directly stored into the ADM EMEM via the Back Bone Bus (BBB). Since the phase-images arrive in bursts of four images each, the memory space for four pictures is reserved in the ADM EMEM.

  The XAM EMEM is mainly used to temporarily buffer image data for UDP streaming. Additionally, it is used to share image data between the CPUs, since the LMU RAM cannot fit a full 16 bit image. For example, CPU 1 saves its output image to the XAM EMEM and then notifies CPU 2 to fetch that image and continue the image processing.
- **PSPR:**
  Frequently called code is placed into the PSPR for fast code execution. Thus, most of the image processing routines are located in this memory.
- **DFlash:**
  The DFlash is used to store the amplitude reference image. It is saved there permanently and does not have to be renewed each time the PFlash is programmed. This memory is not shown in the picture since it is only read and the data is loaded into the DSPR prior to its use, to achieve higher speed.
- **PFlash:**
  The PFlash contains most of the program code and some read-only data. This memory is also not shown in the figure.

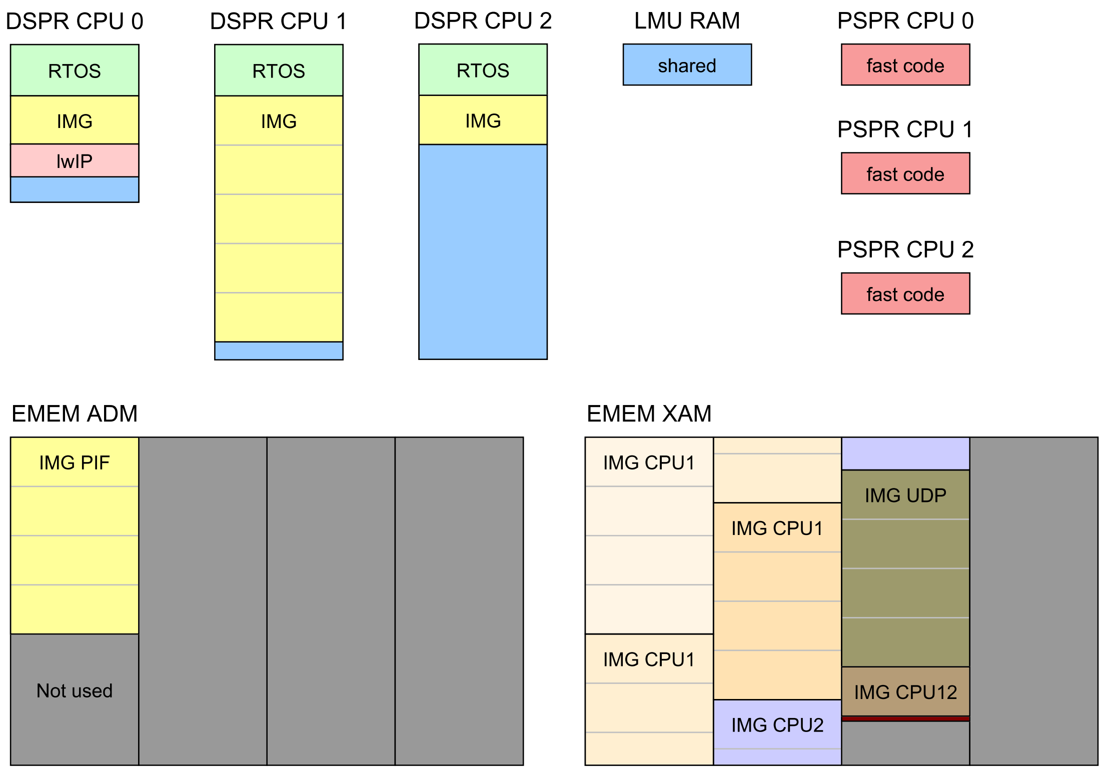


Figure 4.12: Memory configuration.

4.4.4 UDP Data Buffering

To detach the UDP sending task from any other data dependencies, all relevant data is copied to a UDP buffer area in XAM EMEM before any data is sent. Since pipelining is used, the other CPUs are processing new image data already when CPU 0 is ready to send debug data. Hence the data from CPU 1 and CPU 2 has to be buffered to have it still available when the UDP transmission of CPU 0 becomes active. CPU 1 buffers three times the debug data size, and CPU 2 buffers two times the size. The used buffer slot is determined using a running image number. Figure 4.13 shows the buffer concept used to achieve that.

When CPU 0 has finished the decision-making, the debug data of all three CPUs is copied to a separate UDP buffer. This buffer has the purpose to decouple the UDP sending task from all other data dependencies. This makes sense since the UDP sending task on CPU 0 has a low priority and might exceed time constraints. In that case, CPU 0 will recognize that the UDP task has not finished, skip the transmission of the debug data of the current image and resume sending the previous one. To improve the sending performance, data chunks of the UDP buffer are copied into a smaller buffer in the DSPR of CPU 0 before they are actually sent.

To avoid the waste of processing time with copying data to and from the buffer, the Direct Memory Access (DMA) controller of the AURIX is used when suitable.

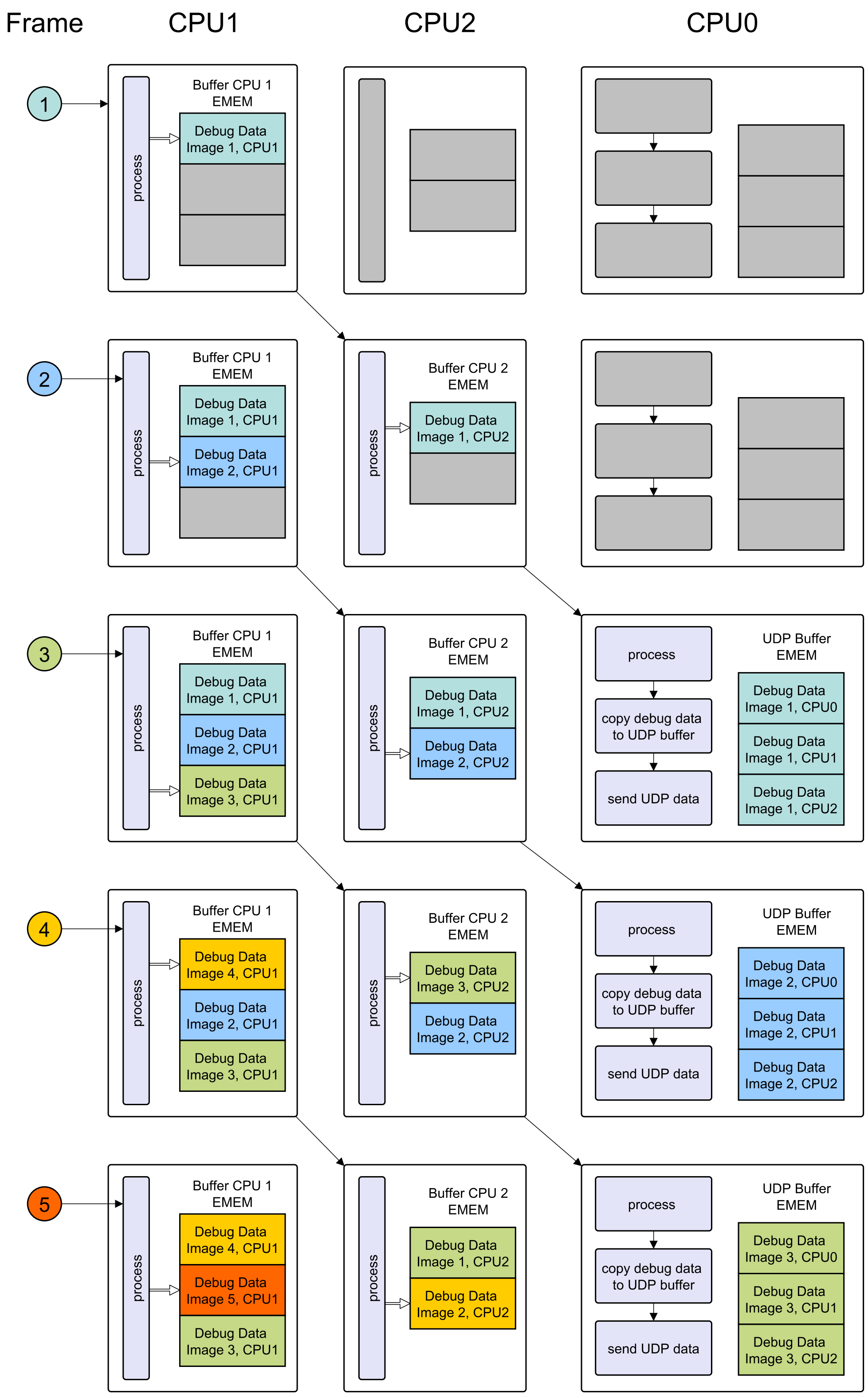


Figure 4.13: Buffering of debug data, to allow pipelining.

4.5 ToF Processing Algorithm

This section describes the implementation details of the processing algorithm on the AURIX, starting with the calculation of the distance image. Afterwards, the pre-processing steps and the object detection is introduced before the section is closed with the description of the decision-making algorithm.

4.5.1 Distance Image Calculation

To calculate the distance image, the microcontroller first receives a sequence of four raw phase-images from the ToF camera board. The CIF of the AURIX microcontroller is configured to automatically save received images to the ADM EMEM. After four phase-images are received, an interrupt on CPU 1 is triggered that unblocks a task copying the image from the EMEM to the local DSPR of CPU 1.

Afterwards, the task starts the conversion of the values from the four captured phase-images into a real and an imaginary component. The phase difference is obtained by calculating the arctangent function of that complex number, using a LUT containing 256 elements and linear interpolation. Before the LUT is used, it is loaded into the DSPR of CPU 1 in order to speed-up the calculation. The offset compensation and the unambiguous range shift are performed directly with the phase difference data, right before the distance data is calculated.

Additionally to the distance data, the amplitude could be calculated from the raw data as well. But to avoid the square root and save computation time, only the squared value is calculated. After this step, the distance image contains values between 0 mm and 8823 mm and is available for further processing.

The processing steps performed are shown in Algorithm 4.1. The calculation of the arctangent is the main part of this algorithm. Thus, this step is described in further detail.

Algorithm 4.1: Distance image calculation on CPU 1.

```
image[0..3] = getPhaseImages( )                          ▷ copy phase-images from EMEM
for every pixel $idx$ do
    $im$, $re$ = CalculateImRe(image[0..3])
    $\Delta\varphi$ = Atan2($im$, $re$)
    $\Delta\varphi$ = GlobalOffsetComp($\Delta\varphi$)
    $\Delta\varphi$ = UnambRangeShift($\Delta\varphi$)
    $d[idx] = \Delta\varphi \cdot rangeFactor$           ▷ unprocessed distance image
    $A^2 = (re * re + im * im)/4$                        ▷ squared amplitude
    ...                                                  ▷ pre-processing starts here
end for
```

Arctangent Computation

The arctangent function is a high-cost function in software and thus is implemented using a LUT with linear interpolation. As seen in Figure 4.14, this approach (Lerp) outperforms two existing implementations using series expansion (libc and Ifx_atan2Q15) and an available CORDIC implementation on the AURIX.

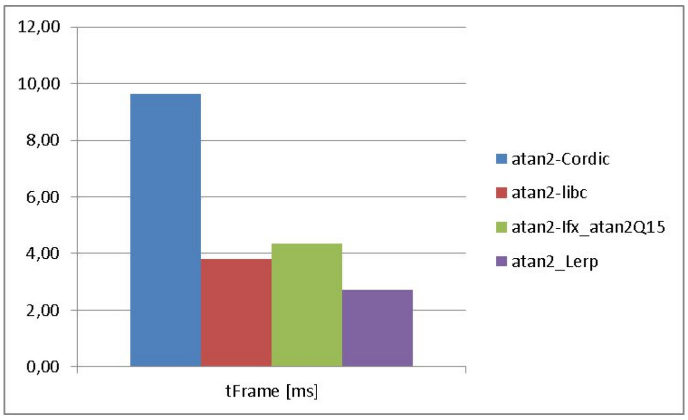


Figure 4.14: Comparison of different arctangent algorithms on the AURIX [Inf14].

Since the real part and imaginary part are known, the four-quadrant arctangent function `atan2(im,re)` is implemented. Caused by the properties of complex numbers, a rotation into the first octant can be performed to simplify the calculation (Figure 4.15a). The rotation angle has to be kept track of, and added to the result afterwards.

The rotation into the first octant is performed as follows:

- **Rotate by π:**
  Points from quadrant 3 and 4 ($Im < 0$) are rotated into quadrant 1 and 2 with a rotation angle of π (Figure 4.15b). To perform the rotation, the real and imaginary part are updated with:

$$x_{new} = -x$$
$$y_{new} = -y$$

- **Rotate by $\frac{\pi}{2}$:**
  Points from quadrant 2 ($Im < 0$) are rotated into quadrant 1 with a rotation angle of $\frac{\pi}{2}$ (Figure 4.15c). To perform the rotation, the real and imaginary part are updated with:

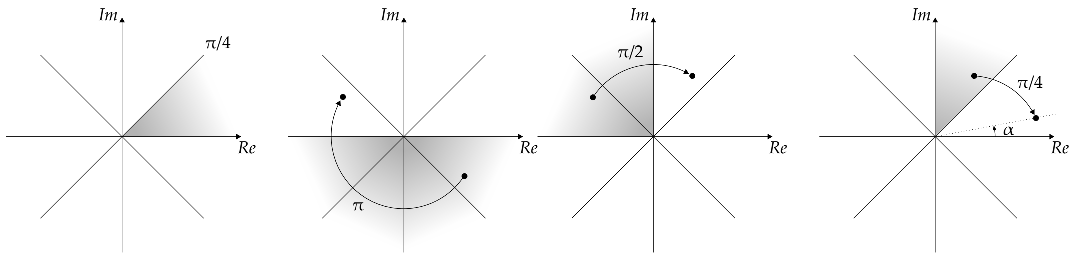
$$x_{new} = y$$
$$y_{new} = -x$$

- **Rotate by $\frac{\pi}{4}$:**
  Points from quadrant 1 with $y > x$ (octant 2) are translated into octant 1 (Fig-

ure 4.15d) with an angle shift of $\frac{\pi}{4}$. To perform this translation, the real and imaginary part are updated with:

$$x_{new} = y - x$$
$$y_{new} = x + y$$

This point translation to achieve a rotation of $\frac{\pi}{4}$ is explained in detail in Appendix A.3. As seen in Figure 4.15d, the length of the vector alters during this translation. But since only the angle is of interest, this does not matter.



(a) First octant. (b) Rotate 180°. (c) Rotate 90°. (d) Rotate 45°.

Figure 4.15: Arctangent calculation simplification.

After the rotation into the first octant, the input domain of the `atan2(im,re)` function is reduced to the range $0 \leq \frac{y}{x} \leq 1$, resulting in an angle $0 \leq$ `atan2(y,x)` $\leq \frac{\pi}{4}$. This assignment is mapped to a LUT containing 256 elements. Additionally, linear interpolation between the two resulting LUT entries is applied, to improve the accuracy of the calculation. After the calculation of the angle within the first octant, the result is back-projected to the original octant to obtain the actual phase difference $\Delta\varphi$.

4.5.2 Pre-Processing

The pre-processing of the distance data is also done on CPU 1. The single steps performed in the pre-processing task are shown in Algorithm 4.2. The first pre-processing steps are performed in the same loop as the distance calculation, to save computation time.

Discarded pixels during pre-processing steps are set to a special value outside the range (12500 mm) to indicate that they are not valid. After pre-processing, the output image is transferred to the XAM EMEM and an interrupt on CPU 2 is triggered to indicate that new data is ready. Important implementation details of the pre-processing steps are further explained.

Camera Angle Conversion

The cosine functions of Equation 3.15 are implemented using a look-up table for all possible angle values. The LUT has 81 entries for angle values between 0° and 40° in steps of 0.5°. The LUT is copied into the DSPR of CPU 1 during the start-up in order to speed-up the execution.

Algorithm 4.2: Pre-processing on CPU 1.

```
for every pixel idx do                                          ▷ distance calculation loop
    ...
    $d[idx]$ = CameraAngleConv($d[idx]$)
    $d_1[idx]$ = AmpThresh1($d[idx]$, $A^2$)
    $d_1[idx]$ = AmplThresh2($d_1[idx]$, $A^2$)
end for
for every pixel $idx$ do                                        ▷ common neighborhood loop
    $d_2[idx]$ = CommonNeighborhood($d[idx]$)
    $d_1[idx]$ = Combine($d_1[idx]$, $d_2[idx]$)
end for
for every pixel $idx$ do                                        ▷ median filter loop
    $d[idx]$ = Median($d_1[idx]$)
end for
```

Amplitude Thresholding

Since the squared amplitude value is available, the comparison value has to be squared as well. The amplitude reference image was obtained using Matlab via the UDP stream of the phase-images. The squared amplitude image is defined as a constant array in the AURIX code and is located in the DFlash. Before usage, it gets copied into the DSPR of CPU 1 to allow faster access.

Another method, without the need to save an additional amplitude image is to calculate an approximation of the amplitude reference image on the AURIX. This approach is described in Appendix A.5

Common Neighborhood Thresholding

CN thresholding is an area operation, since it takes the local neighborhood as input to determine the output for the original pixel. Thus, the method cannot be applied in-place and an additional memory region has to be used for the output.

To save computing time, border pixels are ignored since additional exception handling would significantly increase the computation time.

Median Filter

For the median filter, an efficient way to sort the neighborhood (nine values) is implemented in form of a sorting network. A sorting network consists of a fixed number of statements and implements sorting algorithm with minimum overhead. The implementation of the sorting network can be found in Appendix A.8. Since the discarded values are sorted as well using this method, the median index of the array has to be adapted depending on the number of discarded pixels within the neighborhood. The median filter is also an area operation, and thus has to use a separate memory region for the output. Due to the same reasons as for the CN filter, border pixels are ignored.

4.5.3 Object Detection

Object detection is done on CPU 2 and the implementation details are presented here. When CPU 1 finishes pre-processing, an interrupt is triggered in CPU 2, which unblocks the object detection task running on CPU 2. The task copies the pre-processed image from the XAM EMEM to the DSPR of CPU 2 and performs the object detection steps.

The single steps performed during the object detection are shown in Algorithm 4.3, and the major implementation details of the single steps are explained in this section.

After object detection, the object list is stored in the LMU RAM, and an interrupt in CPU 0 is triggered to initiate further processing.

Algorithm 4.3: Object detection on CPU 2.

```
for every pixel idx do
    $d[idx]$ = AreaOfInterest($d[idx]$)
end for
$h$ = CreateHistogram($d$)
$objectList$ = HistogramSegementation($h, d$)
```

Area of Interest

Trigonometric functions are used to check whether a pixel value is within the AOI or not (see Equation 3.20) . Since the angle values for every pixel are assumed to be constant, the distance limit for every pixel is stored in a LUT in order to save computation time. A LUT for the horizontal distance limits containing 81 values, and a LUT for the vertical distance limits containing 61 values are defined in the AURIX program code. They are loaded into the DSPR of CPU 2 before usage.

Histogram Segmentation

The local minima and very small values of the histogram are evaluated as segmentation borders. To be used as a segmentation border, they have to additionally meet the following criteria:

- A certain amount of pixels within two segmentation borders has to be exceeded.
- The peak between two minima has to exceed a certain height.
- The relation between the minima and the maxima in between has to be within a specified range.
- The two minima should not be too different from each other.

If a potential segmentation border does not meet the criteria, it is either ignored, or marked as a new segmentation border if the histogram value is very small. This avoids pixels to be segmented into objects within low-level histogram areas, which generally do not contain relevant object data.

After the segmentation borders are determined, the pixels within the segmented ranges are assumed to be objects. Next, all pixels of the distance image are iterated through in order to obtain the characteristics (limits, mean values, amount of pixels) of each object.

4.5.4 Decision-Making

If a new object list is available, an interrupt on CPU 0 is triggered and the new data is copied from the LMU RAM to the local DSPR of CPU 0. If a new car state is received via CAN, an interrupt on CPU 0 is triggered and the car state contained in the CAN message data is saved to the DSPR of CPU 0. The decision-making task is unblocked, either if new data is available, or if there was not any new data received within a certain period of time.

The part of the decision-making part of the algorithm, where the decision to brake or not is made, is shown in Algorithm 4.4.

Algorithm 4.4: Decision-making on CPU 0.

```
stopping_dist = CalculateStoppingDistance(car_state)
for every object in object list do
    min_area = CalcMinArea(object.dist)
    if  object.dist < 1.5 stopping_dist and
        (object.area > min_area or object.area > 750) and
        object.area > 10 and
        car_state.current_speed > 500 and
        car_state.forward = true then
            StopVehicle( )
    end if
end for
```

The *stopping_distance* used in the decision-making algorithm has to take into account that the captured distance image was already captured a certain time ago. According to Section 3.4.1 the stopping distance also has to take the processing time T_{proc} and the brake delay time $T_{brake,delay}$ into account.

For a frame rate of 30 FPS ($T_{fps} = 33\,\text{ms}$) the reaction time T_{react}, from the time a frame was captured until the brake is applied, calculates as shown in Equation 4.1. Similar as in Section 3.4.1, fully utilized pipelining is assumed.

$$\begin{aligned} T_{react} &= T_{proc,CPU2} + T_{proc,CPU1} + T_{proc,CPU0} + T_{brake,delay} \\ &= 4 \cdot T_{FPS} + 0.1\,\text{s} \\ &= 233\,\text{ms} \end{aligned} \tag{4.1}$$

Thus, the stopping distance *stopping_distance* is calculated as seen in Equation 4.2. The current speed of the car v and a friction factor $\mu = 0.3$ for an indoor PVC floor are used.

$$\begin{aligned} s_{stop} &= s_{react} + s_{brake} \\ &= v \cdot T_{react} + \frac{v^2}{2\mu g} \end{aligned} \tag{4.2}$$

A criterion for the system to apply emergency braking is that the distance to the detected object is smaller than 1.5 times the calculated stopping distance. This assures that the vehicle is able to avoid the collision.

Another criteria, is that the object exceeds a distance depending area, or a certain area value (i.e., 750 pixels). This distance depending area (min_area) is about the size of a 10 cm x 10 cm square, resulting in a different amount of pixels depending on the distance.

Additionally, the object area has to be higher than a certain minimum (10 pixels) in order to avoid incorrect breaking due to noise. The emergency breaking only takes action if the scaled vehicle is in forward driving mode.

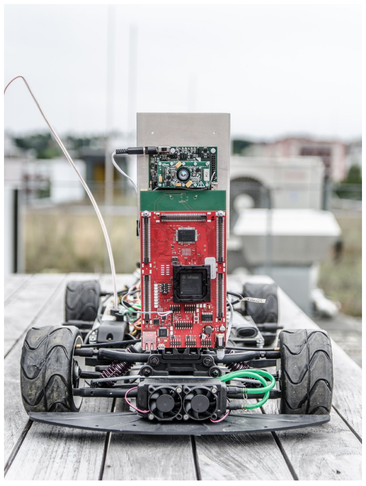
Chapter 5

Results

This chapter is about the actual performance of the system. First the composition of the final prototype is shown. Afterwards the quality of the ToF images is compared for different parameters. Finally, the performance of the used algorithm is discussed regarding computation time and robustness.

5.1 Built-up Platform

This section shows the composition of the built-up platform. A picture of the scaled vehicle with the ToF processing platform mounted in front-view direction is shown in Figure 5.1a. Figure 5.1b shows the top view of the scaled vehicle. The different modules of the scaled vehicle are clearly visible from that perspective. A test scene used to perform different evaluations in order to get comparative results is presented. Additionally, different ToF camera positions are compared and discussed.


(a) Front view.

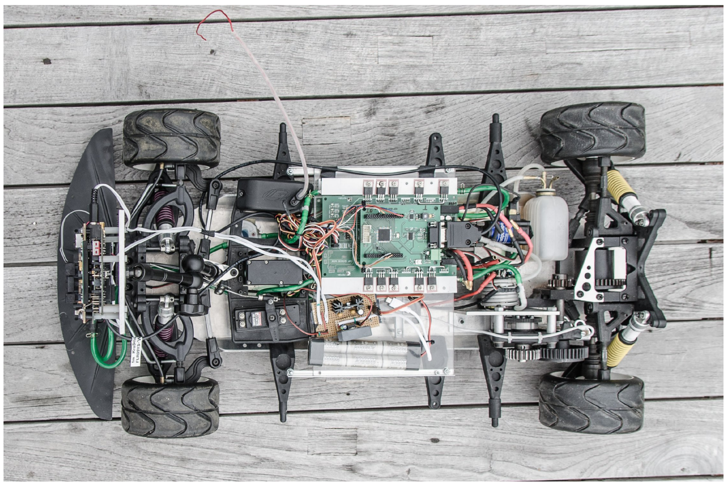
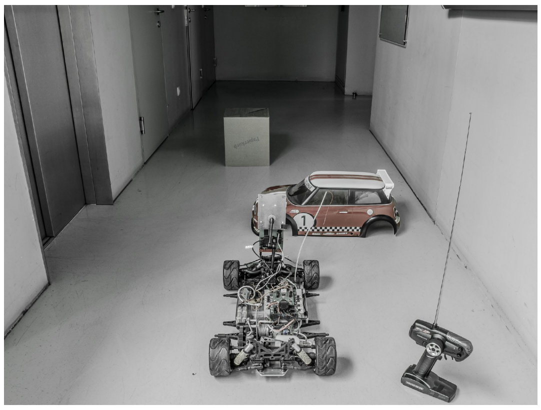

(b) Top view.

Figure 5.1: Scaled vehicle with mounted camera.

5.1.1 Test Scene

An example scene was set up in order to obtain comparable sensor data. A similar setup was built both indoors and outdoors in order to evaluate the influence of sunlight (see Section 5.3.1). The scene contains the chassis of a 1/5 scaled vehicle about 0.5 m and a box approximately 1.5 m from the lens.


(a) Indoor test scene.

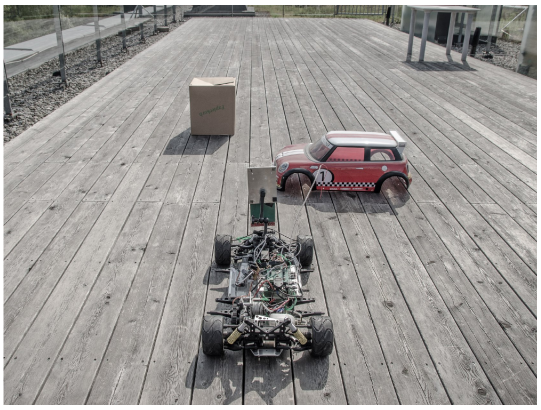

(b) Outdoor test scene.

Figure 5.2: Used test scenes.

5.1.2 Camera Mount

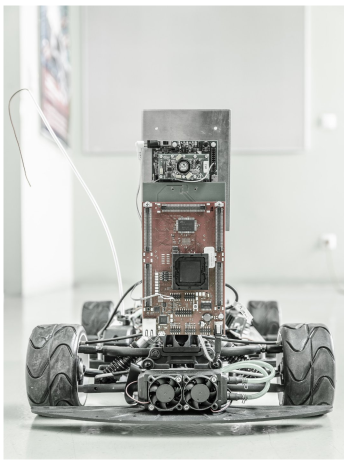
The camera mount is important for the quality of the obtained image. In this thesis the ToF camera is used in front-view direction. So the major variable is the vertical position of the camera and the angle it is pointing to. The Figures 5.3 and 5.4 show three different camera positions with the related distance images. In order to achieve the lower positions of the camera, the processing system had to be rotated. The distance images in Figure 5.4b and Figure 5.4c were rotated back in software in order to obtain a common orientation. If the camera is placed close to the ground, the image quality decreases due to effects such as light scattering and overexposure At a height of about 20 cm, the impact of those errors is almost void.

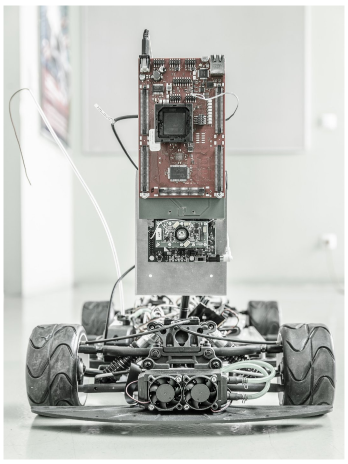
5.2 ToF Camera Configuration

This section deals with the comparison of different illumination times and different modulation frequencies.

5.2.1 Illumination Time

The obtained image quality strongly depends on the choice of the illumination time. The distance images in Figure 5.5 show the same scene, obtained with different illumination


(a) 30 cm above ground.


(b) 20 cm above ground.

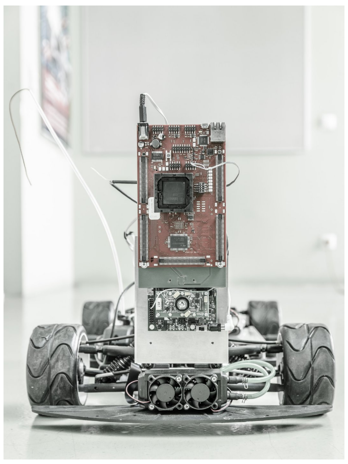
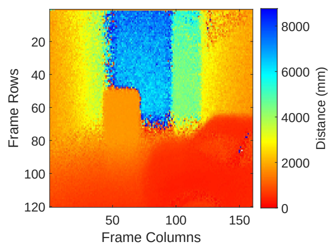
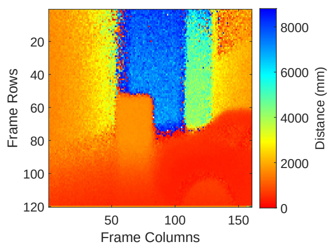

(c) 10 cm above ground.

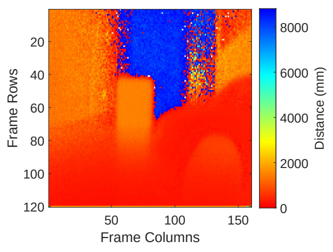
Figure 5.3: Different ToF camera positions.



(a) 30 cm above ground.



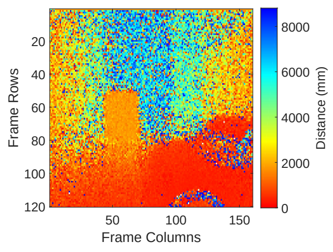
(b) 20 cm above ground.



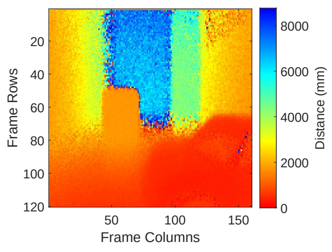
(c) 10 cm above ground.

Figure 5.4: Resulting distance images for different camera positions.

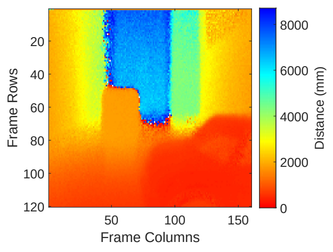
times. A short illumination time results in a noisy image because for many pixels, not enough photons are acquired to gain a robust value (see Figure 5.5a). In contrast, a high illumination time leads to a higher certainty of the single pixel values (see Figure 5.5c). But since a high illumination time also increases the vulnerability to motion blur and saturation, an illumination time of 2 ms is used in this work.



(a) 0.3 ms.



(b) 2 ms.

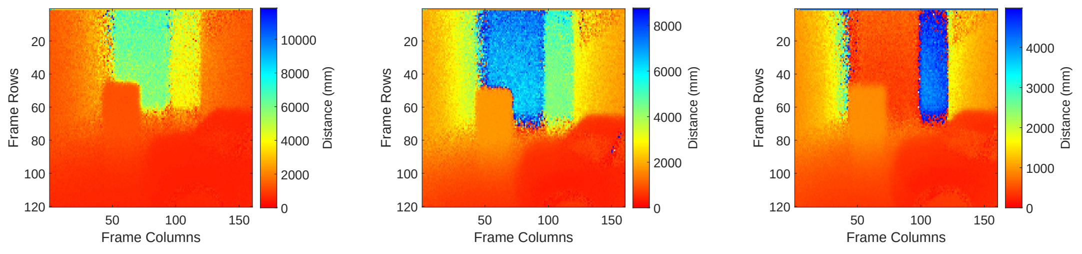


(c) 5 ms.

Figure 5.5: Distance image for different illumination times.

5.2.2 Modulation Frequency

The modulation frequency primarily determines the unambiguous range of the distance image. Higher frequencies result in a lower unambiguous range but in a higher accuracy within that range. Figure 5.6 shows the obtained distance image for different modulation frequencies. At a modulation frequency of 30 MHz, the unambiguous range is 5 m, causing invalid depth values for distances higher than 5 m. This effect is called phase-wrapping, and is clearly visible in Figure 5.6c. In this work, a modulation frequency of 17 MHz is used resulting in an unambiguous range of about 8.3 m.



(a) 12 MHz. (b) 17 MHz. (c) 30 MHz.

Figure 5.6: Distance image for different modulation frequencies.

5.3 Performance

In this section the performance of the whole setup is evaluated in regards to speed and quality.

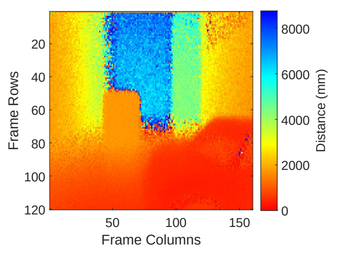
5.3.1 Ambient Light

The performance of the camera was evaluated for different light conditions. This was done by setting up a similar test scene outdoors (see Figure 5.2b) and recording 3D data in order to determine the influence of sunlight.

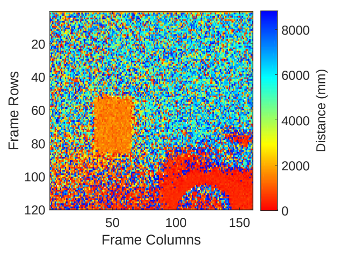
Figure 5.7 and 5.8 show the distance and amplitude image with different intensities of ambient light. The indoor scene, with almost no ambient light present, results in a very good image quality (see Figure 5.7a and 5.8a). As seen in the figures, the noise level increases tremendously in the outdoor scene. The back-light suppression of the ToF sensor is not sufficient for robust outdoor usage. Thus, the platform is only used indoor in this work.

5.3.2 Image Processing Steps

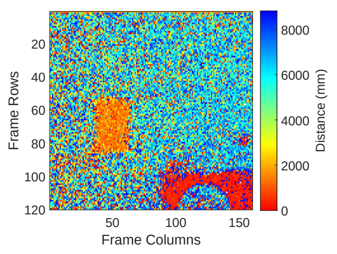
This section gives an example of the full image processing lifespan. The indoor test scene (Figure 5.2a) was used to record and process the 3D data.



(a) Indoor.

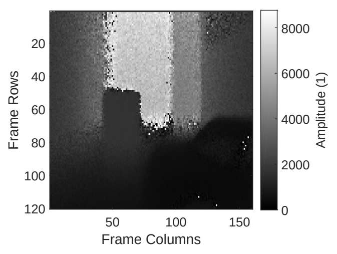


(b) Outdoor, clouded.

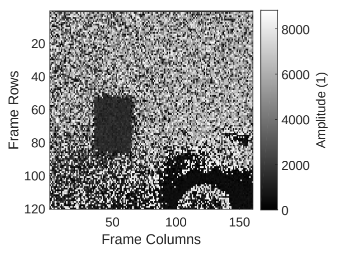


(c) Outdoor, direct sunlight.

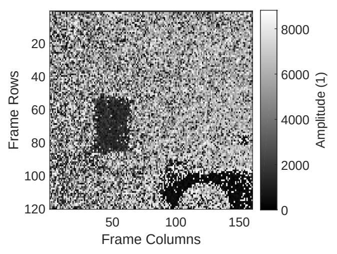
Figure 5.7: Distance image for different light conditions.



(a) Indoor.



(b) Outdoor, clouded.



(c) Outdoor, direct sunlight.

Figure 5.8: Amplitude image for different light conditions.

Four Phase-Images

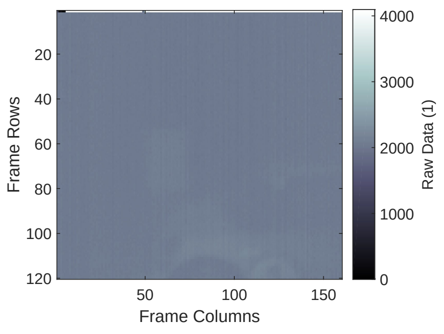
Figure 5.9 shows the raw sensor data received from the ToF evaluation kit, the four phase-images. The first row looks different because pseudo data is transmitted here.

Amplitude and Distance Image

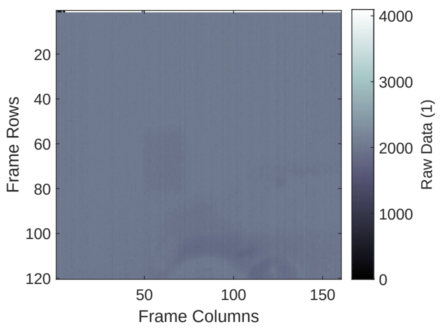
With the four phase-images the distance and amplitude image are calculated. The calculated distance and amplitude image from the phase-images can be seen in Figure 5.10. As seen in the figure, the distance image is not further processed and still contains the global offset. In the final algorithm, the amplitude image is not fully computed on the AURIX. Only the squared amplitude values are calculated and temporarily stored for further processing.

Pre-Processing Steps

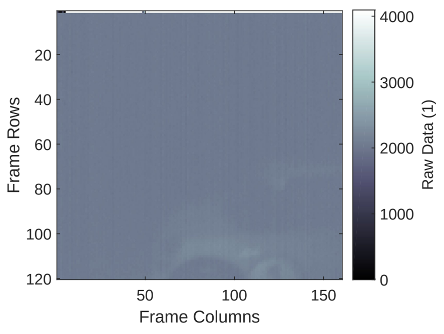
Starting with the unprocessed distance data, multiple pre-processing steps are performed to improve the suitability of the image for object detection. Figure 5.11 shows intermediate images during the single pre-processing steps. Non confident pixels (for example because of a low amplitude value) are discarded and displayed as white pixels.



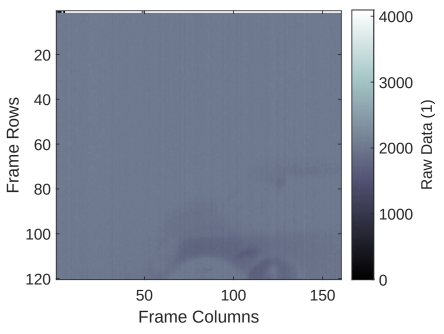
(a) 0°.



(b) 90°.

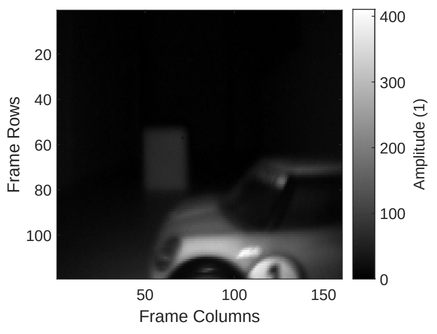


(c) 180°.

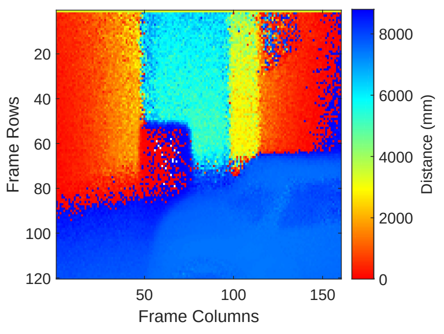


(d) 270°.

Figure 5.9: Raw data (4 phase-images) from ToF evaluation kit.



(a) Amplitude image.

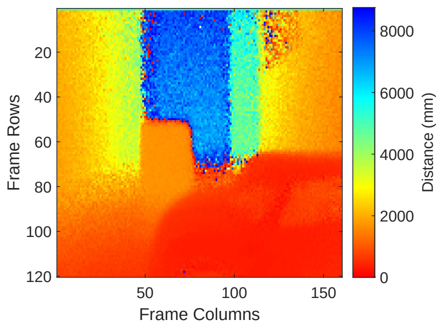


(b) Unprocessed distance image.

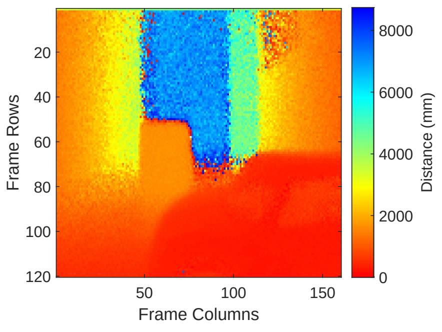
Figure 5.10: Amplitude and distance image.

Object Detection

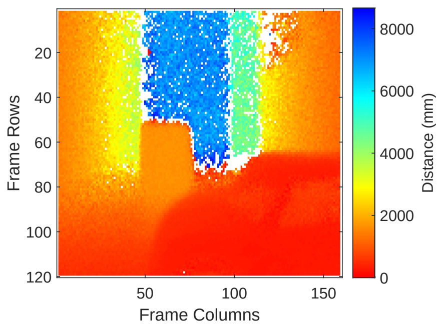
The remaining points within the AOI are shown in Figure 5.12a. Pixels outside the AOI are discarded and displayed as white pixels. Figure 5.13 shows a smoothed version of the histogram from that distance image. This histogram is used for the segmentation of objects at different distances. The detected objects that use the histogram approach are outlined with red rectangles in the distance image seen in Figure 5.12b.



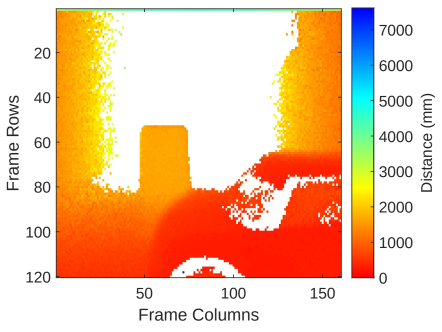
(a) Offset compensation.



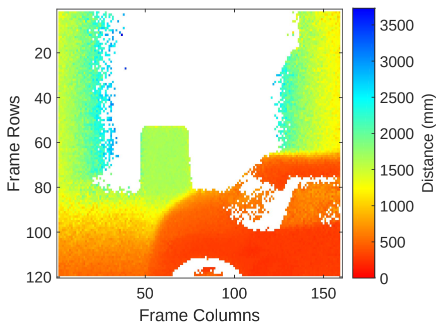
(b) Camera angle conversion.



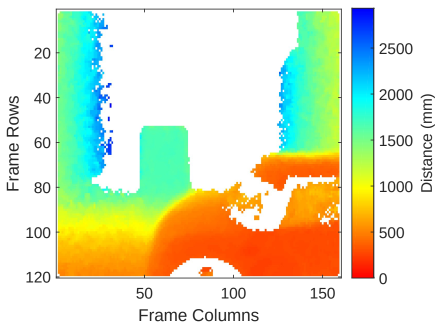
(c) CN filtering.



(d) Amplitude thresholding



(e) Combined data.



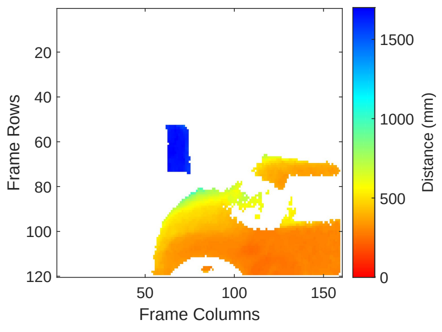
(f) Median filter.

Figure 5.11: Single image pre-processing steps.

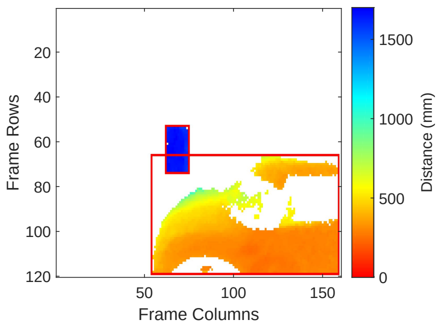
The implemented object detection is simple and efficient. The used histogram segmentation is likely to result in under-segmentation, because distinct objects at the same distance are not separated any further. Since only the pixels within the AOI are considered, only a very limited number of objects are recognized.

5.3.3 Emergency Braking

A very basic version of emergency braking was implemented during this thesis. If an obstacle is detected within the AOI, the current speed of the car is considered in order



(a) AOI.



(b) Detected objects.

Figure 5.12: Object detection.

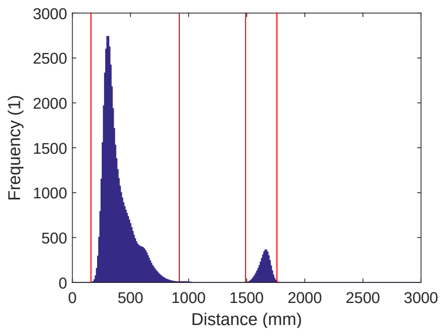


Figure 5.13: Smoothed histogram of the remaining pixels with the segmentation borders marked.

to decide whether a collision is likely to occur or not. If a collision is predicted, the car switches in “CAN Control Mode” and the brakes are applied until the vehicle is stopped. Afterwards the control is given back to the user, and the remote control can be used to move the car. This implementation works very well, when used indoors and with static obstacles.

To predict if a collision is likely to occur, the stopping distance is considered. This distance depends on a number of factors, like the current speed and the road surface. The friction coefficient is defined as a constant in the program code and thus the system cannot adapt to changing road conditions. Thus, the system fails for bad road conditions, if the stopping distance increases.

Other than that, since only the current frame is evaluated and object tracking is not performed, it is not possible to determine the object’s trajectories. Thus, an object moving in the same direction as the car is seen as a static obstacle and the brakes are applied although there is no danger.

Phase-wrapping is still a problem in this implementation, although measures are taken to eliminate it. If a very reflective object outside the unambiguous range is detected, the pre-processing might not be able to discard those pixels. The system might then consider the object located at a wrong distance, and mistakenly apply the brakes.

5.3.4 Time Behavior

The performance regarding processing time on the AURIX is examined. Furthermore, the CPU utilization of the final system is analyzed and the maximum frame rate is determined.

CPU Utilization

The timing of the implemented algorithm on the AURIX is shown in Figure 5.14. The frame rate for this example is about 30 FPS. The utilization of every CPU can be directly seen in the timing diagram.

The timing behavior was recorded with a logic analyzer while processing a camera frame from a real-world scene. The pre-processing algorithm running on CPU 1 is not data dependent and thus the execution time is always the same. The execution time of the algorithm on CPU 2 depends on the number of objects detected, so the processing time is data-dependent. Thus, the time surplus was chosen to be high enough to meet the deadlines in any case. The half-transparent colored time-frames indicate the UDP transmission task on CPU 0. Since it does not provide any functionality to the processing system, it can be totally disabled if no debug data is desired.

Each very short processing pulse on CPU 0 indicates the execution of the decision-making task. The task is executed frequently as the car state and values from the remote control are also transmitted very frequently. It is important that the decision-making algorithm has a short execution time and is immediately activated when new data is available. Since the UDP task running on CPU 0 is not high-priority, it can be interrupted by the decision-making task.

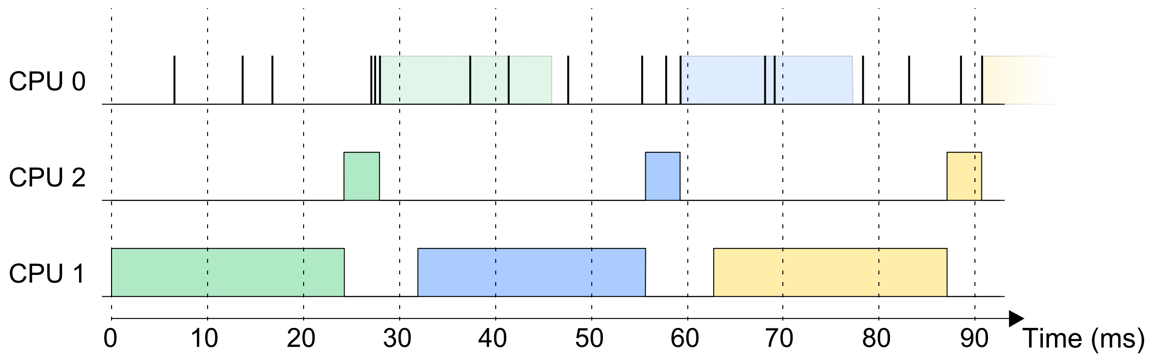


Figure 5.14: Timing diagram of the algorithm.

Frame Rate

According to Section 3.4.2, the frame rate of the camera has to exceed 25 FPS in order to successfully brake before an object. Due to the processing time on CPU 1, the frame rate of the implemented algorithm is limited to about 35 FPS. At higher frame rates, deadlines are violated, leading to malfunction of the system. The utilization of CPU 2 is relatively low, leaving space to extend the object detection part of the processing algorithm.

The algorithm could be partitioned more equally on the three CPUs in order to achieve a higher frame rate. With a slightly simplified modification of the current algorithm and the total exclusion of debug data, a frame rate of at least 60 FPS is possible.

Chapter 6

Conclusion and Future Work

This chapter draws conclusions based on the results obtained during the work. Additionally, multiple ideas for possible future works based on the built-up platform and the results of this thesis are presented.

6.1 Conclusion

ToF cameras are a promising technology to be used in the automotive section in the future. The technology has already proved its feasibility for automotive interior applications, such as for human detection on the driver's seat and gesture control. This thesis evaluates the feasibility of a 3D ToF camera for the automotive environment perception. The camera was mounted on a remote-control 1/5 scale vehicle, pointed in the front direction. An automotive microcontroller processes the ToF data in order to obtain information about objects in front of the vehicle.

As presented in Chapter 5, different positions of the ToF camera on the scaled vehicle were evaluated in order to obtain the most suitable view for the use case. Furthermore, the influence of the illumination time and the modulation frequency on the output image quality were examined and discussed within this work.

The efficient working range of the ToF camera was examined to be limited to around 5 m. Although different pre-processing approaches were applied to enhance the performance, robust object detection could not be achieved for higher distances. The range might be sufficient to provide emergency braking on a relatively slow RC car, but definitely too low to use it on a real world vehicle moving at high speed.

Interfering sunlight causes the quality of the ToF data to decrease drastically. The integrated backlight suppression is not sufficient to avoid negative effects on image quality caused by conditions with direct sunlight. Thus, the built system is not feasible to be used outdoor, due to its limited illumination power.

A basic emergency braking was implemented and successfully tested in an indoor environment. The implemented processing algorithm uses simple and efficient steps to detect objects in front of the vehicle in order to allow a very responsive system. The algorithm was reasonably partitioned onto the three cores of the automotive microcontroller. Due to a sophisticated pipelining approach, the system can handle frame rates up to 30 FPS.

The performance of the used ToF sensor is not sufficient to be used for unrestricted emergency braking on a fully sized vehicle. Still, applications for ToF sensors are possible in special use cases limited to low speeds (e.g., reverse driving assistance or parking assistance). A ToF camera could also be used as a redundant system to ultrasonic sensors. If the ToF sensor technology is further improved, and major limitations such as the range and the vulnerability to sunlight are repealed, ToF cameras might play an important role in automotive vehicle environment perception.

6.2 Future Work

The novel platform, built throughout the progress of this thesis, can be used as a prototype for similar approaches in the future or serve as a base for future projects. This section introduces various ideas for future works based on this thesis.

Alternative processing unit: The AURIX microcontroller is not designed to perform computative costly image processing tasks. To reach a feasible frame rate, low-complexity algorithms and diverse approximations were used. A more efficient object detection could be achieved by replacing the AURIX microcontroller by a high-performance system.

Instead of replacing the automotive microcontroller, an additional processing unit could be interposed. Doing so, all computative expensive processing steps could be moved to the high performance chip.

Another application is to use an additional platform to make the debug data available via a wireless stream. The single-board computer can be connected to the AURIX TriBoard via Ethernet and forward the UDP data-stream via the wireless interface.

Custom system: The used setup consists of multiple evaluation boards, feasible for prototyping. The single components made the system bulky and inflexible, and caused signal propagation problems. A possible improvement would be the construction of a Printed Circuit Board (PCB) with all required components on a single board.

Additional sensors: In the current setup, the scaled vehicle is equipped with the rotation-speed sensor and the ToF camera only. To obtain a more accurate state of the car and its surroundings, additional sensors such as radar, lidar, ultrasonic or 2D cameras could be implemented. The combination of data from the different sensors would allow a more accurate object detection, although increasing the computational cost.

Special use cases: Some application-specific use cases can be implemented using the current setup. The system could be extended to applications such as automatic distance control or parking assistance, to simulate real world use cases with the scaled vehicle. Another interesting use case would be to follow an object with the scaled vehicle.

Appendix A

Technical Additions

This chapter contains some details of the work worth mentioning, but either too detailed or not directly relevant to the outcome of the work itself.

A.1 ToF Module Connection

Different approaches were made to establish a connection between the ToF camera module and the AURIX TriBoard. The different methods and the reasons why a flexible connection is not practicable, are presented here.

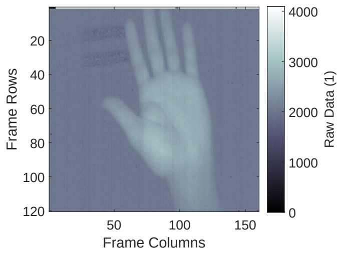
Flexible Connection

A flexible connection between the AURIX TriBoard and the ToF evaluation kit was established using a flexible ribbon cable.

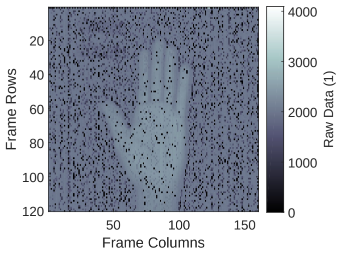
Unfortunately, this setup leads to erroneous data at the used data rate (pixel clock = 66 MHz). Multiple effects like crosstalk or reflections negatively influence the data quality. Effectively terminating the signals is difficult, since the wave impedance changes during the signal path. Figure A.1 shows a raw phase-image using the alternative adapter board for different ribbon cables. The signal quality is acceptable for very short cables, but for a reasonable cable length, the quality is not sufficient. Since the quality of the calculated distance image adds up the errors of the single phase-images, this flexible setup is not feasible to be used in the targeted environment.

Pixel Clock

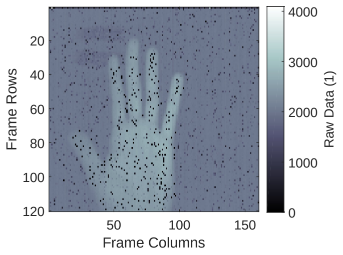
Since the image quality is not acceptable with the full pixel clock, the influence of a a lower pixel clock was evaluated. The pixel clock of PIF on the ToF evaluation kit can be divided by 2, 4 and 8. Unfortunately, there occur some problems with the CIF when using lower settings for the pixel clock. It seems that not every pixel clock edge is recognized, which causes a horizontal blur in the received image on the AURIX. This behavior can be



(a) Short cable, 5 cm.



(b) Long cable, 30 cm.



(c) Long cable with shielding, 30 cm.

Figure A.1: Phase-image 0° of the setup with a flexible cable.

seen in Figure A.2. Thus, the setup is used with the pixel clock configured to the highest possible rate ($f_{clk} = 66\,\text{MHz}$).

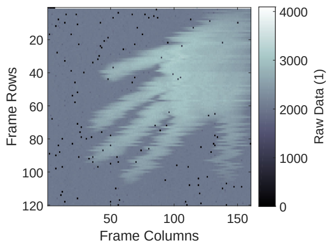


Figure A.2: Phase-image 0°, at a pixel clock of 8.33 MHz.

A.2 Voltage Converter Board

An additional voltage converter board using a step down converter (LMR14050, Texas Insruments) was designed during this work. The board can be used to efficiently convert the available battery voltage to supply the electronic components of the car. Figure A.3 shows the schematic, and Figure A.4 shows the layout of the voltage converter board. The PCB was created using the electronic design automation software EAGLE [Eag15].

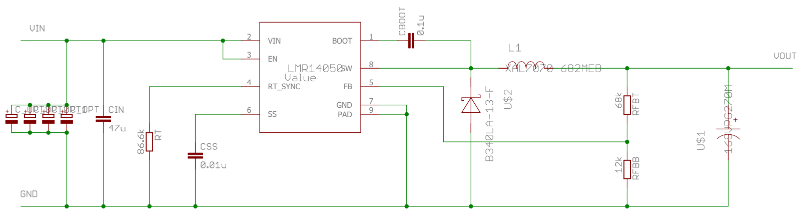


Figure A.3: Voltage regulator schematic.

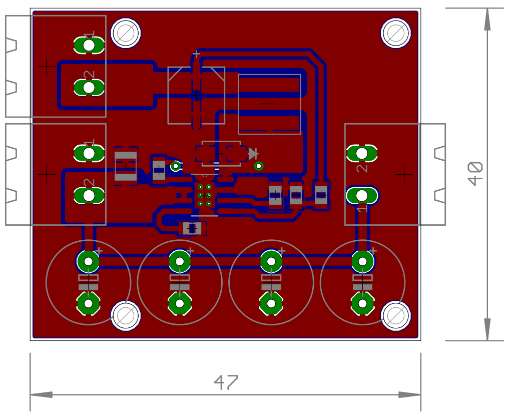


Figure A.4: Voltage regulator PCB.

A.3 Rotation into the First Octant

For the arctangent calculation process, the complex point is rotated into the first octant of the complex plane. This is done to reduce the domain of input values and make the use of a LUT feasible. Figure A.5 shows the step performed to rotate a point by $\frac{\pi}{4}$. The components of the destination point P' are calculated with:

$$x' = y_1 - x_1$$
$$y' = x_1 + y_1$$

As seen in the figure, an isosceles, right-angled triangle arises, which causes an angle of $\alpha = \frac{\pi}{4}$ between the points P_1 and P'.

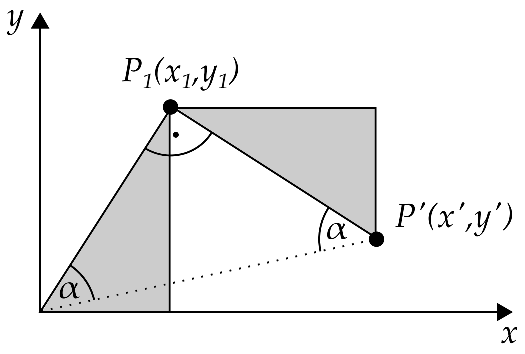


Figure A.5: Angle rotation in the first quadrant.

A.4 CAN Message Structure

The most important parts of a CAN message are the 11 bit identifier defining the priority of the message and the 0-8 bytes of data.

The car-status messages are split up into two messages which are sequentially sent. The structure of the two messages and their contents is shown in Table A.2 and Table A.3. All CAN message IDs used in the system are described in Table A.1. Every time new values from the remote are received by the car's microcontroller, a message with the current remote values is sent via the CAN interface. The structure of that message is described in Table A.4.

The structure of the CAN message to take over the control is described in Table A.5. The control remains active until a message with the CAN master ID, not containing the magic number, is sent (as seen in Figure 4.6).

Alias	ID	comment
CAN Device ID1	0xA0	car status 1
CAN Device ID2	0xA1	car status 2
CAN Device DBG	0xDB	remote values
CAN Master ID	0x50	CAN take-over

Table A.1: CAN IDs.

Byte	Content	Comment
0 1	AutoMotorCurrent	Motor Current in mA
2 3	AutoBattVoltage	Battery Voltage in mV
4 5	AutoPower	Power in mW
6 7	AutoRotationSpeed	Rotation speed in mm/s

Table A.2: CAN message structure: CAN Device ID1.

A.5 Amplitude Reference Image

An approach to make the amplitude reference image available on the AURIX is to calculate an approximation of it. A good approximation of the amplitude reference image can be calculated with:

$$A_{ref}(x, y) = 10 + 135 \cdot cos(|x - 59| \cdot 0.5°)^7 \cdot cos(|y - 79| \cdot 0.5°)^7$$

The Figure A.6 shows the surface plot of the measured and calculated amplitude reference image. The calculated version is similar enough to the measured version, to use it for the processing algorithm in this work. To make the calculation of the amplitude reference feasible on the AURIX, the function $cos(x \cdot 0.5°)^7$ is made available using a LUT for all integer values of x between 0 and 80.

Byte	Content	Comment
0 1	AutoMotorVoltage	Motor Voltage in mV
2	AutoTemp1	Temperature
3	AutoTemp2	Temperature
4	AutoTemp3	Temperature
5	AutoTemp4	Temperature
6	AutoTemp5	Temperature
7	AutoReserved	Reserved

Table A.3: CAN message structure: CAN Device ID2.

Byte	Content
0 1	Remote Ch1 Value
2 3	Remote Ch2 Value
4 5	Remote Ch3 Value

Table A.4: CAN message structure: CAN Device DBG.

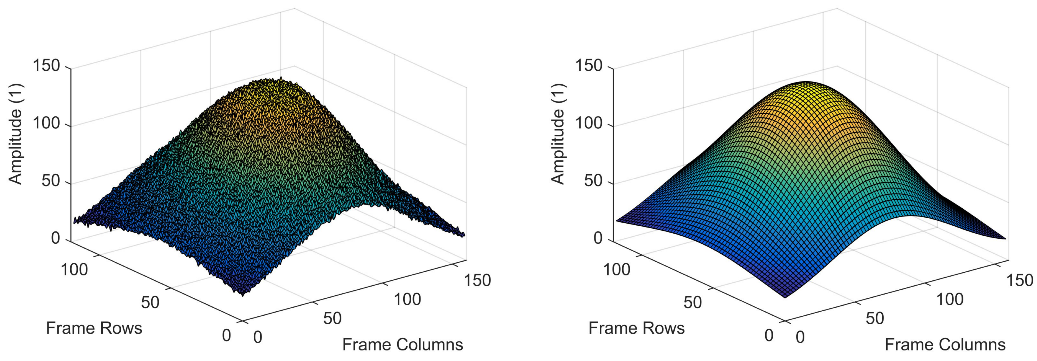
Byte	Content	Comment
0	Auto Mode	Mode for driving (forward/backward/brake)
1 2	Auto Value	Value for that mode
3	Steering Mode	Mode for steering (left/right)
4	Steering Value	Value for that mode
5	-	Reserved for future use
6	0xCA	“Magic number”
7	0xFE	“Magic number”

Table A.5: CAN message structure: CAN Master ID.

A.6 Additional Tools

PCAN View

PCAN View [Pea15] is a tool providing the functionality to send and receive CAN messages with a PC. To use the program, a special CAN to USB adapter is required. The application can be used to display received CAN messages, or to send certain CAN messages to the



(a) Measured. (b) Calculated approximation.

Figure A.6: Amplitude reference image, measured and calculated approximation.

embedded system.

USBee Suite

The USBee Suite [Cwa16] is the software that comes with the USBee Test Pod, a USB device with multiple input channels. The device provides the functionality of a logic analyzer. With the software, it is possible to display and analyze the captured signals. For debugging, certain pins of the AURIX can be connected to this oscilloscope. This could also be done with a regular logic analyzer, but this tool is smaller and can be directly used with the development workstation.

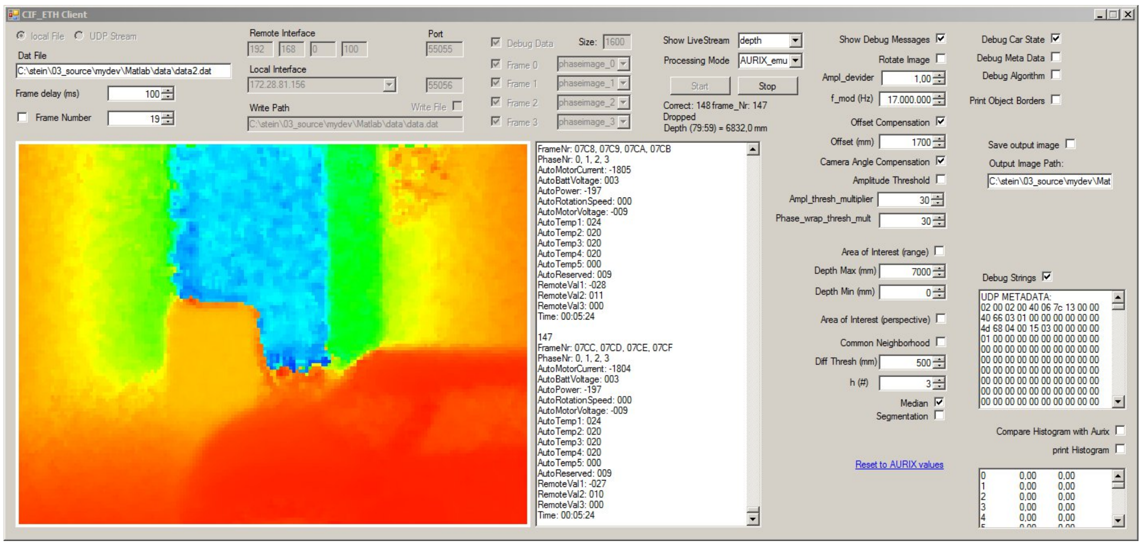
A.7 PC Debug Application

This section gives a description of the application developed in Visual Studio to support the AURIX development process. Figure A.7 shows a screenshot of the application.

Input Source

- Live stream:
  Live data from the AURIX can be streamed via Ethernet and directly viewed in this application. The streaming data can include images, meta-data etc. If live-streaming is selected, it is also possible to save the whole sequence for later use.

- Saved sequence:
  Previously saved sequences can be used as input. In this mode it is possible to set the FPS or evaluate single frames.



local File [✓] UDP Stream
Dat File: C:\stein\03_source\mydev\Matlab\data\data2.dat
Frame delay (ms): 100
Frame Number ☐ 19
Remote Interface: 192 168 0 100 Port: 55055
Local Interface: 172.28.81.156 55056
Write Path: C:\stein\03_source\mydev\Matlab\data\data.dat Write File ☐
Debug Data ☑ Size: 1600
Frame 0 ☑ phaseimage_0
Frame 1 ☑ phaseimage_1
Frame 2 ☑ phaseimage_2
Frame 3 ☑ phaseimage_3
Show LiveStream: depth
Processing Mode: AURIX_emu

```
Correct: 148 frame_Nr: 147
Dropped
Depth (79:59) = 6832,0 mm
```

```
FrameNr: 07C8, 07C9, 07CA, 07CB
PhaseNr: 0, 1, 2, 3
AutoMotorCurrent: -1805
AutoBattVoltage: 003
AutoPower: -197
AutoRotationSpeed: 000
AutoMotorVoltage: -009
AutoTemp1: 024
AutoTemp2: 020
AutoTemp3: 020
AutoTemp4: 020
AutoTemp5: 000
AutoReserved: 009
RemoteVal1: -028
RemoteVal2: 011
RemoteVal3: 000
Time: 00:05:24

147
FrameNr: 07CC, 07CD, 07CE, 07CF
PhaseNr: 0, 1, 2, 3
AutoMotorCurrent: -1804
AutoBattVoltage: 003
AutoPower: -197
AutoRotationSpeed: 000
AutoMotorVoltage: -009
AutoTemp1: 024
AutoTemp2: 020
AutoTemp3: 020
AutoTemp4: 020
AutoTemp5: 000
AutoReserved: 009
RemoteVal1: -027
RemoteVal2: 010
RemoteVal3: 000
Time: 00:05:24
```

Show Debug Messages ☑
Rotate Image ☐
Ampl_devider: 1,00
f_mod (Hz): 17.000.000
Offset Compensation ☑
Offset (mm): 1700
Camera Angle Compensation ☑
Amplitude Threshold ☐
Ampl_thresh_multiplier: 30
Phase_wrap_thresh_mult: 30
Area of Interest (range) ☐
Depth Max (mm): 7000
Depth Min (mm): 0
Area of Interest (perspective) ☐
Common Neighborhood ☐
Diff Thresh (mm): 500
h (#): 3
Median ☑
Segmentation ☐

Debug Car State ☑
Debug Meta Data ☐
Debug Algorithm ☐
Print Object Borders ☐
Save output image ☐
Output Image Path: C:\stein\03_source\mydev\Mat
Debug Strings ☑

```
UDP METADATA:
02 00 02 00 40 06 7c 13 00 00
40 66 03 01 00 00 00 00 00 00
4d 68 04 00 15 03 00 00 00 00
01 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
```

Compare Histogram with Aurix ☐
print Histogram ☐

```
0    0,00    0,00
1    0,00    0,00
2    0,00    0,00
3    0,00    0,00
4    0,00    0,00
```

Figure A.7: Screenshot of the application.

Data Mode

- Debug data:
  If debug data is included, the size has to be stated here in order to have correct alignment of the other data.

- Image data:
  Then the structure of the transmitted image frames has to be selected. It is possible to select between 0 and 4 frames. Examples for the image data: the four phase-images or one single distance image.

- Processing mode:
  These modes define what kind of processing the application performs on the transmitted data. There is an emulation mode that performs calculations very similar to the AURIX platform (same types, approximations etc.). Then there is an exact calculation mode that uses no approximations in order to estimate the impact of certain approximations. The availability of certain processing modes depends on the type of the included frames.

Camera Parameters

The application performs a similar data-processing as the implementation on the AURIX. To evaluate and control the intermediate results, it is possible to activate only certain steps of the processing. Additionally, it is possible to modify certain parameters of the algorithm during operation. This makes it very convenient to tweak the parameters, especially when used with saved real-world sequences.

Output

- Image data:
  The displayed image depends on the processing mode. It can be the original received image data from the AURIX or the output of the processing in this application. The application can only display amplitude and distance data, other image data is not supported.
- Debug data:
  Additionally, there are multiple text-boxes available that can be filled with various debug data. This data can include running numbers, sensor data, execution time, histograms, control variables, meta-data etc. Displaying that data in the PC application is a very good option to debug the AURIX code.

A.8 Sorting Network

To perform median filtering, the neighboring pixel values have to be sorted for every pixel of the image. To minimize the execution time of the median filter, the sorting part has to be implemented very efficiently. Using a sorting network is one way to accomplish that. The implementation of a sorting network of size 9 consists of 25 if-statements and introduces very little overhead compared to the implementation of a generic sorting algorithm. Listing A.1 shows the program code of the sorting network.

A.9 Linker Description File

The linker description file is located in `workspace\ld\iROM.ld` and defines the memory placement of the program code and the variables in the physical memory. Some important details of the linker description file are explained here. The structure of the source folder containing the AURIX code is shown in Figure A.8.

It is desired to map the variables into the corresponding CPU's local DSPR. The code-snippet in Listing A.2 shows the memory configuration within the linker description file.

The DSPR of each CPU is separated into a global part `DMI_DSPR` and an exclusive part, e.g., `DMI_DSPR0_EX`. To achieve the variable mapping separation by the contained directory of the source file, the output section structure shown in Listing A.3 was used.

The wildcard pattern "*" matches any number of characters, while "?" matches a single character only. So the input file wildcard pattern `*src?cpu0*` matches every file in the `cpu0` folder of the `src` directory, e.g., `src\cpu0\main0.c`. The output sections for the other folders are defined as well and thus the exclusive mapping for every CPU is achieved. This is a convenient solution, compared to manually setting a certain input section (e.g., `.dspr0_extended`) for each variable at the time of declaration.

```
if (a[0] > a[1]) { uint16 tmp = a[0]; a[0] = a[1]; a[1] = tmp; }
if (a[3] > a[4]) { uint16 tmp = a[3]; a[3] = a[4]; a[4] = tmp; }
if (a[6] > a[7]) { uint16 tmp = a[6]; a[6] = a[7]; a[7] = tmp; }
if (a[1] > a[2]) { uint16 tmp = a[1]; a[1] = a[2]; a[2] = tmp; }
if (a[4] > a[5]) { uint16 tmp = a[4]; a[4] = a[5]; a[5] = tmp; }
if (a[7] > a[8]) { uint16 tmp = a[7]; a[7] = a[8]; a[8] = tmp; }
if (a[0] > a[1]) { uint16 tmp = a[0]; a[0] = a[1]; a[1] = tmp; }
if (a[3] > a[4]) { uint16 tmp = a[3]; a[3] = a[4]; a[4] = tmp; }
if (a[6] > a[7]) { uint16 tmp = a[6]; a[6] = a[7]; a[7] = tmp; }
if (a[0] > a[3]) { uint16 tmp = a[0]; a[0] = a[3]; a[3] = tmp; }
if (a[3] > a[6]) { uint16 tmp = a[3]; a[3] = a[6]; a[6] = tmp; }
if (a[0] > a[3]) { uint16 tmp = a[0]; a[0] = a[3]; a[3] = tmp; }
if (a[1] > a[4]) { uint16 tmp = a[1]; a[1] = a[4]; a[4] = tmp; }
if (a[4] > a[7]) { uint16 tmp = a[4]; a[4] = a[7]; a[7] = tmp; }
if (a[1] > a[4]) { uint16 tmp = a[1]; a[1] = a[4]; a[4] = tmp; }
if (a[2] > a[5]) { uint16 tmp = a[2]; a[2] = a[5]; a[5] = tmp; }
if (a[5] > a[8]) { uint16 tmp = a[5]; a[5] = a[8]; a[8] = tmp; }
if (a[2] > a[5]) { uint16 tmp = a[2]; a[2] = a[5]; a[5] = tmp; }
if (a[1] > a[3]) { uint16 tmp = a[1]; a[1] = a[3]; a[3] = tmp; }
if (a[5] > a[7]) { uint16 tmp = a[5]; a[5] = a[7]; a[7] = tmp; }
if (a[2] > a[6]) { uint16 tmp = a[2]; a[2] = a[6]; a[6] = tmp; }
if (a[4] > a[6]) { uint16 tmp = a[4]; a[4] = a[6]; a[6] = tmp; }
if (a[2] > a[4]) { uint16 tmp = a[2]; a[2] = a[4]; a[4] = tmp; }
if (a[2] > a[3]) { uint16 tmp = a[2]; a[2] = a[3]; a[3] = tmp; }
if (a[5] > a[6]) { uint16 tmp = a[5]; a[5] = a[6]; a[6] = tmp; }
```

Listing A.1: Sorting network of size 9.

```
src
├── cpu_all
│   ├── Free RTOS source
│   └── ...
├── cpu0
│   ├── lwIP source
│   ├── main0.c
│   └── ...
├── cpu1
│   ├── CIF source
│   ├── main1.c
│   └── ...
└── cpu2
    ├── main2.c
    └── ...
```

Figure A.8: AURIX source code directory.

```
MEMORY
{
  PMU_PFLASH0 (rx!p): org = 0x80000000, len = 2M
  PMU_PFLASH1 (rx!p): org = 0x80200000, len = 2M
  PMU_PFLASH2 (rx!p): org = 0x80400000, len = 2M
  PMU_PFLASH3 (rx!p): org = 0x80600000, len = 2M
  PMU_DFLASH0 (r!xp): org = 0xAF000000, len = 1M
  PMU_DFLASH0_1 (r!xp): org = 0xAF100000, len = 16K
  PMU_DFLASH1 (r!xp): org = 0xAF110000, len = 64K
  BROM (rx!p): org = 0x8FFF8000, len = 32K
  PMI_PSPR (wx!p): org = 0xC0000000, len = 32K
  DMI_DSPR (w!xp): org = 0xD0000000, len = 20K
  DMI_DSPR0_EX (w!xp): org = 0x70005000, len = 100K
  DMI_DSPR1_EX (w!xp): org = 0x60005000, len = 220K
  DMI_DSPR2_EX (w!xp): org = 0x50005000, len = 220K
  LMU_SRAM (w!xp): org = 0x90000000, len = 32K
}
```

Listing A.2: Memory configuration.

```
.dspr0_extended   :
{
  *(.dspr0_extended)
  *(.dspr0_extended*)
  *src?cpu0*(.bss)
  *src?cpu0*(.bss*)
  *src?cpu0*(.sbss)
  *src?cpu0*(.sbss*)
  *src?cpu0*(.bbss)
  *src?cpu0*(.bbss*)
  *src?cpu0*(.zbss)
  *src?cpu0*(.zbss*)
} > DMI_DSPR0_EX /* DMI_DSPR0_EX: Local Data RAM (DSPR0) Extended - CPU 0 */
```

Listing A.3: Output section description.

A.10 ToF Camera Configuration

The following code-snippet shows the values of the camera configuration registers. These values are sent to the camera, before the continuous image capturing is started. The initialization data can be found in the file `workspace\h\cif\Mira_regs.h`.

```
const uint16 init_data[INIT_LENGTH][2] =
{
{ANAIP_SENSEN, 0x00FF},
{ANAIP_GPIOMUX5, 0x0180},
{ANAIP_GPIOMUX7, 0x018D},
{ANAIP_PADGPIOCFG0, 0x1515},
{ANAIP_PADGPIOCFG1, 0x1515},
{ANAIP_PADGPIOCFG2, 0x1515},
{ANAIP_PADGPIOCFG3, 0x1515},
{ANAIP_PADGPIOCFG4, 0x1515},
{ANAIP_PADGPIOCFG5, 0x1515},
{ANAIP_PADGPIOCFG6, 0x1315},
{ANAIP_PADGPIOCFG7, 0x1513},
{ANAIP_PADGPIOCFG8, 0x1515},
{ANAIP_PADGPIOCFG9, 0x0415},
{ANAIP_PADGPIOCFG10, 0x0404},
{ANAIP_PADGPIOCFG11, 0x0004},
{ANAIP_ADCBG1, 0x0003},
{ANAIP_ADCEN, 0xFFFF},
{ANAIP_ADCRESET, 0x0001},
{ANAIP_PIXIFEN, 0xFFFF},
{ANAIP_PIXREFBGEN, 0x0003},
{ANAIP_PIXREFEN, 0x0FFF},
{ANAIP_PLLBGEN, 0x0001},
{ANAIP_PSPADCFG, 0x1513},
{ANAIP_VMODREG, 0x0007},
{CFGCNT_S00_EXPOTIME, 0x109A}, // 2ms @17MHz
{CFGCNT_S00_FRAMERATE, 0x0000},
{CFGCNT_S00_PS, 0x0000},
{CFGCNT_S00_PLLSET, 0x0002},
{CFGCNT_S01_EXPOTIME, 0x109A},
{CFGCNT_S01_FRAMERATE, 0x0000},
{CFGCNT_S01_PS, 0x0888},
{CFGCNT_S01_PLLSET, 0x0002},
{CFGCNT_S02_EXPOTIME, 0x109A},
{CFGCNT_S02_FRAMERATE, 0x0000},
{CFGCNT_S02_PS, 0x0444},
{CFGCNT_S02_PLLSET, 0x0002},
{CFGCNT_S03_EXPOTIME, 0x109A},
{CFGCNT_S03_FRAMERATE, 0x1000},  // ca 30 FPS
{CFGCNT_S03_PS, 0x0CCC},
{CFGCNT_S03_PLLSET, 0x0002},
{CFGCNT_TRIG, 0x0000},
{CFGCNT_STATUS, 0x0000},
{CFGCNT_CSICFG, 0x0080},
{CFGCNT_PIFCCFG, 0x1050},
{CFGCNT_PIFTCFG, 0x0000},
{CFGCNT_BINCFG, 0x0005}, //2x2 binning
{CFGCNT_ROICMINREG, 0x0010}, //center 160x120 with 2x2 binning
{CFGCNT_ROICMAXREG, 0x014F},
{CFGCNT_ROIRMINREG, 0x0018},
{CFGCNT_ROIRMAXREG, 0x0107},
{CFGCNT_ROS1, 0x006},
{CFGCNT_ROS2, 0x0060},
{CFGCNT_IFDEL, 0x4088},
{CFGCNT_CTRLSEQ, 0x0003},
{CFGCNT_EXPCFG1, 0x0380},
{CFGCNT_EXPCFG2, 0x000F},
{CFGCNT_EXPCFG3, 0x1E0A},
{CFGCNT_EXPCFG4, 0x1C05},
{CFGCNT_PSOUT, 0x0313},
{CFGCNT_PLLCFG1_LUT1, 0x466D}, // 12MHz
{CFGCNT_PLLCFG2_LUT1, 0xEC4F}, // 12MHz
{CFGCNT_PLLCFG3_LUT1, 0x0BC4}, // 12MHz
{CFGCNT_PLLCFG1_LUT2, 0x2241}, // 30MHz
{CFGCNT_PLLCFG2_LUT2, 0x13B2}, // 30MHz
{CFGCNT_PLLCFG3_LUT2, 0x03BB}, // 30MHz
{CFGCNT_PLLCFG1_LUT3, 0x6449}, // 17MHz
{CFGCNT_PLLCFG2_LUT3, 0x2762}, // 17MHz
{CFGCNT_PLLCFG3_LUT3, 0x03F6}, // 17MHz
{CFGCNT_PLLCFG1_LUT4, 0x4059},
{CFGCNT_PLLCFG2_LUT4, 0xC4ED},
{CFGCNT_PLLCFG3_LUT4, 0x13CE},
{MTCU_POWERCTRL, 0x1408},
{PIF_PIFHSIZE, 0x00A0},
{PIF_PIFVSIZE, 0x0078}
};
```

List of Abbreviations

ADM	Application Data Memory
AOI	Area of Interest
AURIX	Automotive Realtime Integrated NeXt Generation Architecture
BBB	Back Bone Bus
CAN	Controller Area Network
CARE	Community Database on Accidents on the Roads in Europe
CIF	Camera Interface
CMOS	Complementary Metal–Oxide–Semiconductor
CN	Common Neighborhood
CORDIC	Coordinate Rotation Digital Computer
CPU	Central Processing Unit
CSI-2	Camera Serial Interface-2
DC	Direct Current
DFlash	Data Flash
DMA	Direct Memory Access
DSP	Digital Signal Processing
DSPR	Data Scratch-Pad RAM
EMEM	Extended Memory
FPGA	Field-Programmable Gate Array
FPS	Frames per Second
GiST	Generalized Search Tree
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
HSYNC	Horizontal Synchronization
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
LMU	Local Memory Unit
LUT	Look-up Table
lwIP	Lightweight IP Stack

MSB	Most Significant Bit
NiMH	Nickel–Metal Hydride
PCB	Printed Circuit Board
PFlash	Program Flash
PIF	Parallel Interface
PIXCLK	Pixel Clock
PLL	Phase-Locked Loop
PMD	Photonic Mixing Device
PSPR	Program Scratch-Pad RAM
RAM	Random-Access Memory
RC	Radio-Controlled
RF	Radio Frequency
RISC	Reduced Instruction Set Computing
ROI	Region of Interest
RTOS	Real-Time Operating System
SAE	Society of Automotive Engineers
SDK	Software Development Kit
SNR	Signal-to-Noise Ratio
SVM	Support Vector Machine
TCP/IP	Transmission Control Protocol/Internet Protocol
TOF	Time-of-Flight
TV	Total Variation
UDP	User Datagram Protocol
VSYNC	Vertical Synchronization
XAM	Extended Application Memory

Bibliography

[Ada11] Adam Dunkels. *LwIP, lightweight IP*. Version 1.3.2. 2011. URL: http://savannah.nongnu.org/projects/lwip/.

[Atm08] Atmel Corporation. *8-bit AVR Microcontroller with 32K/64K/128K Bytes of ISP Flash and CAN Controller*. AT90CAN128. Rev. 7679H–CAN–08/08. Atmel. 2008.

[Bam15] Bamji, C. S., O'Connor, P., Elkhatib, T., Mehta, S., Thompson, B., Prather, L. A., Snow, D., Akkaya, O. C., Daniel, A., Payne, A. D., et al. "A 0.13 μm CMOS system-on-chip for a 512× 424 time-of-flight image sensor with multi-frequency photo-demodulation up to 130 MHz and 2 GS/s ADC". In: *IEEE Journal of Solid-State Circuits* 50.1 (2015), pp. 303–319.

[Bel00] Bellis, S. J. and Marnane, W. P. "A CORDIC Arctangent FPGA Implementation for a High-Speed 3D-Camera System". In: *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing: 10th International Conference, FPL 2000 Villach, Austria, August 27–30, 2000 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 485–494.

[Böh09] Böhme, M., Haker, M., Riemer, K., Martinetz, T., and Barth, E. "Face Detection Using a Time-of-Flight Camera". In: *Dynamic 3D Imaging: DAGM 2009 Workshop, Dyn3D 2009, Jena, Germany, September 9, 2009. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 167–176.

[Cwa16] CWAV, Inc. *USBee Suite*. Version 1.1.75. 2016. URL: http://www.usbee.com.

[Dal14] Dalbah, Y., Rohr, S., and Wahl, F. M. "Detection of dynamic objects for environment mapping by time-of-flight cameras". In: *2014 IEEE International Conference on Image Processing (ICIP)*. Oct. 2014, pp. 971–975.

[Dru15] Druml, N., Fleischmann, G., Heidenreich, C., Leitner, A., Martin, H., Herndl, T., and Holweg, G. "Time-of-Flight 3D imaging for mixed-critical systems". In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. July 2015, pp. 1432–1437.

[Eag15] CadSoft Computer. *EAGLE, Einfach Anzuwendender Grafischer Layout Editor*. Version 7.5.0. 2015. URL: https://cadsoft.io/.

[Eur15] The European Commission. *Annual Accident Report 2015*. 2015.

[Far06] Fardi, B., Dousa, J., Wanielik, G., Elias, B., and Barke, A. "Obstacle Detection and Pedestrian Recognition Using A 3D PMD Camera". In: *2006 IEEE Intelligent Vehicles Symposium*. 2006, pp. 225–230.

[Fis10] Fischer, J., Seitz, D., and Verl, A. “Face Detection using 3-D Time-of-Flight and Colour Cameras”. In: *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*. June 2010, pp. 1–5.

[Gan10] Ganapathi, V., Plagemann, C., Koller, D., and Thrun, S. “Real time motion capture using a single time-of-flight camera”. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. June 2010, pp. 755–762.

[Gok04] Gokturk, S. B., Yalcin, H., and Bamji, C. “A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions”. In: *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW ’04. Conference on*. June 2004, pp. 35–35.

[Han13] Hansard, M., Lee, S., Choi, O., and Horaud, R. *Time-of-Flight Cameras, Principles, Methods and Applications*. Springer London, 2013.

[Hig15] HighTec EDV-Systeme GmbH. *Free TriCore Entry Tool Chain*. Version 1.0. 2015. URL: http://www.hightec-rt.com/en/.

[Hoe13] Hoegg, T., Lefloch, D., and Kolb, A. “Real-Time Motion Artifact Compensation for PMD-ToF Images”. In: *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 273–288.

[Inf13] Infineon Technologies. *Infineon 3D Image Sensor, IRS10x0C*. Product Brief. May 2013.

[Inf14] Infineon Technologies. *MIRA Aurix TC299 Demonstrator*. Version 0.1. Dec. 2014.

[Inf15] Infineon Technologies. *REAL3 image sensor family, 3D depth sensing based on Time-of-Flight*. Product Brief. Nov. 2015.

[Inf16] Infineon Technologies. *AURIX – TC297TA – AURIX family*. Product Brief. Aug. 2016.

[Lan00] Lange, R. “3D time-of-flight distance measurement with custom solid-state image sensors in CMOS/CCD-technology”. PhD thesis. University of Siegen, 2000.

[Lan01] Lange, R. and Seitz, P. “Solid-state time-of-flight range camera”. In: *Quantum Electronics, IEEE Journal of* 37.3 (Mar. 2001), pp. 390–397.

[Len11] Lenzen, F., Schäfer, H., and Garbe, C. “Denoising Time-Of-Flight Data with Adaptive Total Variation”. In: *Advances in Visual Computing: 7th International Symposium, ISVC 2011, Las Vegas, NV, USA, September 26-28, 2011. Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 337–346.

[Len13] Lenzen, F., Kim, K. I., Schäfer, H., Nair, R., Meister, S., Becker, F., Garbe, C. S., and Theobalt, C. “Denoising Strategies for Time-of-Flight Data”. In: *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 25–45.

[Lin10] Lindner, M. “Calibration and Realtime Processing of Time-of-Flight Range Data”. PhD Thesis. Computer Graphics Group, University of Siegen, Dec. 2010.

[Mat15] The Mathworks, Inc. *MATLAB version.* Version 8.5.0.197613 (R2015a). 2015. URL: https://www.mathworks.com.

[Mic12] Microsoft Corporation. *Microsoft Visual Studio Express 2012 for Windows Desktop.* Version 11.0.50727. 2012. URL: https://www.visualstudio.com/.

[Möl05] Möller, T., Kraft, H., Frey, J., Albrecht, M., and Lange, R. “Robust 3D Measurement with PMD Sensors”. In: *In: Proceedings of the 1st Range Imaging Research Day at ETH.* 2005, pp. 3–906467.

[Mur07] Mure-Dubois, J. and Hügli, H. “Real-time scattering compensation for time-of-flight camera”. In: *Proceedings of the ICVS Workshop on Camera Calibration Methods for Computer Vision Systems.* 2007.

[Nat08] Natroshvili, K., Schmid, M., Stephan, M., Stiegler, A., and Schamm, T. “Real time pedestrian detection by fusing PMD and CMOS cameras”. In: *Intelligent Vehicles Symposium, 2008 IEEE.* June 2008, pp. 925–929.

[Pea15] PEAK-System Technik GmbH. *PCAN-View.* Version 4.0.29.426. 2015. URL: http://www.peak-system.com/.

[Pla10] Plagemann, C., Ganapathi, V., Koller, D., and Thrun, S. “Real-time identification and localization of body parts from depth images”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* May 2010, pp. 3108–3113.

[Pla16] Plank, H., Holweg, G., Herndl, T., and Druml, N. “High performance Time-of-Flight and color sensor fusion with image-guided depth super resolution”. In: *2016 Design, Automation Test in Europe Conference Exhibition (DATE).* Mar. 2016, pp. 1213–1218.

[Pmd15] Pmdtechnologies gmbh. *CamBoard pico flexx.* Reference Design Brief. 2015.

[Rea11] Real Time Engineers Ltd. *FreeRTOS.* Version 7.1.0. 2011. URL: http://www.freertos.org.

[Sae14] SAE On-Road Automated Vehicle Standards Committee and others. *Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems.* Technical Report J3016_201401. 2014.

[Sch07] Schamm, T., Vacek, S., Natroshvilli, K., Marius Zöllner, J., and Dillmann, R. “Autonome Mobile Systeme 2007: 20. Fachgespräch Kaiserslautern, 18./19. Oktober 2007”. In: Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Chap. Hinderniserkennung und -verfolgung mit einer PMD-kamera im automobil, pp. 219–225.

[Sch09] Schmidt, M. and Jähne, B. “A Physical Model of Time-of-Flight 3D Imaging Systems, Including Suppression of Ambient Light”. In: *Dynamic 3D Imaging: DAGM 2009 Workshop, Dyn3D 2009, Jena, Germany, September 9, 2009. Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–15.

[Sch11a] Schmidt, M. and Jähne, B. "Efficient and robust reduction of motion artifacts for 3D Time-of-Flight cameras". In: *2011 International Conference on 3D Imaging (IC3D)*. Dec. 2011, pp. 1–8.

[Sch11b] Schmidt, M. "Analysis, Modeling and Dynamic Optimization of 3D Time-of-Flight Imaging Systems". PhD thesis. IWR, Fakultät für Physik und Astronomie, University of Heidelberg, 2011.

[Sch11c] Schwarz, L. A., Mkhitaryan, A., Mateus, D., and Navab, N. "Estimating human 3D pose from Time-of-Flight images based on geodesic distances and optical flow". In: *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*. Mar. 2011, pp. 700–706.

[Sch15] Schockaert, C., Garcia, F., and Mirbach, B. "Guidance image based method for real-time motion artefact handling on Time-of-Flight cameras". In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. June 2015, pp. 1246–1251.

[Uki11] Ukil, A., Shah, V. H., and Deck, B. "Fast computation of arctangent functions for embedded applications: A comparative analysis". In: *2011 IEEE International Symposium on Industrial Electronics*. June 2011, pp. 1206–1211.

[Vol13] Volvo Car Corporation. *Volvo car group initiates world unique swedish pilot project with selfdriving cars on public roads*. [Online; accessed 15-August-2016]. Dec. 2013. URL: https://www.media.volvocars.com/global/en-gb/media/pressreleases/136182/volvo-car-group-initiates-world-unique-swedish-pilot-project-with-self-driving-cars-on-public-roads.

[Vol16] Volvo Car Corporation. *Volvo Car Group, Global Newsroom*. [Online; accessed 27-July-2016]. 2016. URL: https://www.media.volvocars.com/.

[Vol59] Volder, J. E. "The CORDIC Trigonometric Computing Technique". In: *IRE Transactions on Electronic Computers* EC-8.3 (Sept. 1959), pp. 330–334.

[Wal07] Wallhoff, F., Rub, M., Rigoll, G., Gobel, J., and Diehl, H. "Improved Image Segmentation using Photonic Mixer Devices". In: *2007 IEEE International Conference on Image Processing*. Vol. 6. Sept. 2007.

[Wei14] Wei, X., Phung, S. L., and Bouzerdoum, A. "Object segmentation and classification using 3-D range camera". In: *Journal of Visual Communication and Image Representation* 25.1 (2014), pp. 74–85.

[Win09] Winner, H., Hakuli, S., and Wolf, G. *Handbuch Fahrerassistenzsysteme - Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort : mit 550 Abbildungen und 45 Tabellen*. 2009. Aufl. Berlin Heidelberg New York: Springer-Verlag, 2009.