Machine Vision for Industrial Robot Systems

Abstraction and Implementation for "drag & bot"

Master Thesis

Presented to

the **"Hochschule der Medien Stuttgart"**

In Fulfillment of the Requirements for the Degree

Master of Science

by

Alexander Georgescu

Matriculation number: 35268

February 2019

HOCHSCHULE DER MEDIEN

Supervisors:

Prof. Dr. Jens-Uwe Hahn (Hochschule der Medien)

Pablo Quilez Velilla (drag & bot GmbH)

Period: September the 3rd, 2018 until February the 28th, 2019

Declaration of Authenticity

I declare that all material presented in this paper is my own work or fully and specifically acknowledged wherever adapted from other sources. I understand that if at any time it is shown that I have significantly misrepresented material presented here, any degree or credits awarded to me on the basis of that material may be revoked. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief.[1]

Location, Date

Signature

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.[2]

Ort, Datum

Unterschrift

[1] Koc09, S. 2.
[2] Rie02, S. 42.

Abstract

This work proposes and validates a hardware-independent interface for industrial machine vision systems. The proposed interface and methods allow to integrate computer vision applications using different hardware, camera positions and software modules in a common way increasing the compatibility, the ease of use and reducing the integration cost. The interface was validated through the ROS (Robot Operating System) based robot programming framework drag & bot showing different real industrial-relevant examples.

This work first analyses the differences between the existing machine vision and the state of art solutions nowadays including different types of cameras and positions. The focus lies on the company's goals to achieve software that is extendable and reusable.

Then in the main part, the interface and related methods are proposed covering all processes needed for configuring the machine vision system such as for example camera calibration. Finally it describes the implementation of the interface including different prototypes, a "sequence system" that minimizes work required at modifying or expanding functionality. The paper closes with a reevaluation of the final results, taking the initial goals into account and focusing on the conclusions obtained. Furthermore, an outlook to the future of machine vision for drag & bot is provided.

The conclusion that the interface fulfills the requirements resulted in inclusion of the interfaces in the drag & bot software as part of the internal functionality.

Keywords:
machine vision, interface, extensibility, camera, smartcamera, calibration, industrial robot

Zusammenfassung

Diese Arbeit schlägt ein machine vision Interface für das Industrieroboter Kontrollsystem drag & bot vor und begründet es. Das Interface hat zum Ziel hardwareunabhängig zu sein und Ziele der, betreuenden Firma wie Widerverwendbarkeit und Erweiterbarkeit zu erfüllen. Diese werden mit analysierten Lösungen auf dem Stand der Technik mit Fokus auf verschiedenen Kamera-Arten und Nutzer-Anwendungen verglichen.

Der Hauptteil befasst sich mit dem Design und Implementation eines Interfaces für SmartCamera welches Robot Operating System (ROS) - Services einsetzt um die interne Kommunikation abzukapseln. Des weiteren wird die Entwicklung einer Kamera-zu-Roboter Kalibrierung beschrieben welche darüber hinaus ein universelles Sequenz-basiertes Interface umfasst welches den Implementationsaufwand reduziert.

Schlussendlich wird die resultierende Software, insbesondere im Einsatz in Demonstrations Aufbauten analysiert und hinsichtlich der Zielvorgabe verglichen. Das Resultat is dass das Interface ihren Zweck für die drag & bot GmbH mit den momentan vorliegenden Hardwarekomponenten erfüllt.

Stichwörter:
machine vision, Interface, Erweiterbarkeit, Kamera, SmartCamera, Kalibrierung, Industrieroboter

Contents

Chapter 1

Introduction and Analysis

The following sections describe the currently common approaches nowadays at machine vision as well as the motivation for the drag & bot GmbH to carry out this research project and introduce the results of this work in the software.

1.1 Machine Vision

"Machine Vision" refers to hard- and software solutions that use optical sensors like cameras to solve industrial tasks. Common tasks in this field are decision making and the guidance of machines (e.g. industrial robots).[1] The name also refers to the industrial branch that develops, produces and maintains products serving the needs in this field of other companies.
This technology is part of the broader field of computer vision but delimits itself from it by referring to the applications in industrial environments only. That range of applications is the focus of this thesis.

[1] cf. COG, p.3 1-4.

1.1.1 Purpose and Market

Before machine vision appeared in the 1970s, the task of decision-making based on visual information was done directly by humans. Workers were either trained specifically for this task, or performed it as part of their usual duty.[2]. For example, they decided whether a produced workpiece fulfilled a set of desired criteria like shape, color and quality.
Machine Vision has the purpose of replacing special trained personnel or relieve normal personnel off their duty. This way it can increase production speed, production capacity and reduce costs. Furthermore, quality can be increased as well due to higher accuracy of digital systems.
The market for machine vision in Germany increased by 16% to reach €1.9 billion in 2014[3]. It accounts for approximately 40% of the whole European market.[4]

A different source states:

> The global machine vision market is expected to reach USD 15.46 billion by the end of 2022 with 8.18% [growth] during forecast period 2017-2022.
>
> Quote 1: [Market Forecast by Cooked Research Reports Rep17, para. 3]

1.1.2 General Requirements and Constraints

Due to the industrial environment of the hardware and software solutions, special considerations have to be taken into account during development to achieve a competitive and extensible technology and therefore product on the market:

Reliability
Production lines have to work uninterrupted during long period of time and each component like machine vision has to fulfill its purpose during this period.

Usability
The machine vision system may not require more knowledge from interacting personal than needed to reduce the cost for the company. A lower usability results in higher training cost and possibly higher risk of mistakes. A definition and evaluation of usability is needed for a successful product.

Mechanical Stability
The location of use can involve powered machines that induce vibrations and therefore affect the machine vision component.

[2] cf. Jue15, p.3 12-16.
[3] cf. Wen15, para. 7.
[4] cf. Wen15, para. 9.

Temperature Stability
Certain production processes like welding result in high environmental temperatures.

Resistance to Environmental Influences (liquids, steam, sparks etc.)
Industrial machines may require coolants like water or oil which cannot always be contained. Additionally certain tools can emit particles.

Integrability
Communication of the output of the machine vision process with other components of the production facility needs to follow widely accepted industrial standards to achieve integrability.

Direct/Indirect Costs
The product itself and the requirements of additional components needs to be cost efficient to achieve a positive balance in the cost plan of the production facility.

1.2 State of Art

With over four hundred seventy two companies providing solutions for the machine vision market (the number of exhibitors at the largest fair of the industry branch[5]), the number of different approaches would exceed the scope of this work. Therefore the most common and relevant variants are presented and form the base for later evaluation.

1.2.1 2D Cameras

> 2D machine vision rules the machine vision market by accounting for a major chunk (41%) of the overall market.
>
> Quote 2: [Market Forecast by Inkwood Research Res, Key Findings]

As the quote 1.2.1 claims, this type of hardware is the most common in applications. Their main feature is the output of a two dimensional array that contains brightness-values in the case of a monochrome and intensity of the color components for color cameras. This array is provided through an interface to a separate processing computer system.
Some examples for this category of cameras from major manufacturers are: Basler AG, Teledyne Technologies Inc. and SICK[6] as well as their subsidiary companies are shown in detail within the table 1.1. For this work low and medium price range are prioritized to fulfill cost requirements of the target customers of drag & bot GmbH.

[5] cf. Stu18, p.3 12-16.
[6] cf. Res, Main image.

NAME	acA640-90uc	scA1400-30gm	Genie Nano-CL C2420
Manufacturer	Basler	Basler	Teledyne Salsa
Product Range	Basler ace classic	Basler Scout	Genie Nano
Color/Grayscale	Color	Grayscale	Color
Resolution	658 x 492	1392 x 1040	2448 x 2048
Sensor	1/3" CCD, Sony ICX424	2/3" CCD, Sony ICX285	2/3" CMOS, Sony IMX264
Connectivity	USB 3.0, Hardware-Trigger	Gigabit Ethernet, Hardware-Trigger	Camera Link, Hardware-Trigger
SNR	41.7dB	41.3dB	Not public
Price Region	Low-Range (200€+)	Mid-Range	Mid-Range
Remarks	Especially small, 90fps	Claimed as especially versatile	Ethernet version: Nano-GigE

NAME	Genie TS-M1920	IV-HG500CA	I2D601N-MCB71
Manufacturer	Teledyne Dalsa	Keyence	Sick
Product Range	Genie TS	IV	midiCam
Color/Grayscale	Grayscale	Color	Grayscale
Resolution	2048 x 1080	752 x 480	1280 x 1024
Sensor	CMOS, Not public	1/3" CMOS, Not public	1/3" CMOS, e2V
Connectivity	Gigabit Ethernet, Hardware-Trigger	FTP, Ethernet, PROFINET	Gigabit Ethernet
Price Region	High-Range	Low-Range	Mid-Range
Remarks	70fps	Computation unit "IV-HG10" turns into smartcam	Rugged, higher-res versions available

Table 1.1: 2D Cameras

1.2.2 3D Cameras

Three-dimensional (3D) cameras differ from their two-dimensional counterparts by providing depth information. They give "a point cloud (three dimensional coordinates) output. It is a digitized model of location and shape of object(s) scanned"[7]. Their benefits are the capability to perceive objects with a higher degree of information compared to two-dimensional counterparts at the same optical resolution. Additionally because color or brightness are only a part of this information, 3D cameras can be less dependent of reproducible illumination conditions which influence only color and brightness.
Detailed descriptions of the technologies used to achieve 3D information is out of scope of this paper, but the following three variants are described briefly:

Time of Flight (ToF)

Light is sent from a source attached to the camera and it is measured how long the light took to be reflected by the observation area. Knowing the speed of light, it is possible to recompute the total distance from the light source, to the area and to the sensor and that for every pixel[8].

[7] cf. **TM3DM**, p.1, para. 2.
[8] cf. Bas, para. 3.

Structured Light (Shape-from-Shading)

A distinct pattern of light is projected onto the observation area. From the difference of the reflected pattern it is possible to recompute a possible 3D shape of the area[9].
The variation "shape from shading" utilizes different light-sources to detect minor distance changes to the camera and can detect engravings or scratches[10].

Stereo Camera

Two 2D cameras take images simultaneously from different viewpoints. Using pattern-matching algorithms, a possible shape of the observation area can be computed.

As an overview of State of the Art solutions, the table 1.2 shows examples for 3D Cameras from major manufacturers:

NAME	Basler ToF	3D-A5030	SV-M-S1
Manufacturer	Basler	Cognex	Ricoh
Product Range	Basler ToF	3D-A5000	SV-M
Detection Type	Time of Flight	Structured Light	Stereoscopy
Resolution	640 x 480	Not public	2x 1280 x 960
3D Depth Range	0m to 13m	1465mm to 1545mm	800mm to 1200mm
3D Accuracy	+/-1cm	195-200µm	1mm
Price Region	Low-Range	No information	No information
Remarks	Integrated lightsource	Other models of the range have different observation areas	Has external computation box

Table 1.2: 3D Cameras

1.2.3 Smart Cameras

While 2D and 3D cameras like the ones provided above, send raw, uninterpreted image data to the surrounding system that uses machine vision, Smart Cameras are able to perform advanced processing on their own. This means in the most common case, the information that is transmitted from the camera is not an array of pixel values but prepared information that can be used to automatically make decisions or influence the behavior of other components of the system.
Smart Cameras can provide not only recognition and localization of objects but also verify the quality, perform measurements or extract information like color or shape. Furthermore, they support standardized features like reading bar and QR codes.
The table 1.3 shows examples for Smart Cameras from major manufacturers as well as a regional manufacturer (Balluff):

[9] cf. Coga, cpt. 2.
[10] cf. KEY, p.10 LumiTrax Verarbeitung.

NAME	BVS002C	CA-HX048C + CV-X100	IN-SIGHT 2000-110
Manufacturer	Balluff	Keyence	Cognex
Product Range	BVS	CV-X	2000
Sensor	1/1.8" CMOS, monochrome	1/3" CMOS, color	1/3" CMOS, monochrome
Resolution	1280 x 1024	784 x 596	640 x 480
Access System	Integrated Web Server	Customizable Software	Software
Programming/ Teaching-System	Procedural drag and drop 'applications'	Advanced assistant system (customizable)	Advanced assistant system
Major Features	Localization, Verification, measurements, Barcode/Text reader	Filters, Defect-Detection, Localization, Auto-Teach, Optional Shape-from-Shading	Quality-Check, Counting, Pattern-Detection
Connectivity	GigE, IOs	GigE, IOs	GigE, IOs
Price Region	3300€	Not public	Low Range
Remarks	Was the major focus during the development of this paper	Camera: CA-HX048C Compute module: CV-100 Camera-to-Robot calibration	The -1300 and 130C models have measurement features

Table 1.3: Smart Cameras

1.2.4 Computation Power

When machine vision is performed without a Smart Camera like mentioned above, the computing system that receives the uninterpreted camera data requires a significant amount of calculations to extract desired information.
Three major approaches exist:

1.2.4.1 CPU Based

An approach is the usage of a Central Processing Unit (CPU) like it is found in desktop as well as industrial computers.
The advantage lies in the universality of CPUs which allows to implement the algorithms that process images in a large number of programming languages; including scripting languages like Python[11].
The disadvantage is the higher requirement of resources compared to specialized processing systems like the following.

1.2.4.2 GPU Based

Graphic cards are designed to process hundreds or thousands of minor tasks in parallel. By splitting the algorithm into parallelizable steps it is possible to achieve a higher computation power at same cost compared to CPUs[12]. Another advantage is the scalability because multiple GPUs can be used on one computer.
A disadvantage is the requirement of dedicated hardware (usually including a computer that contains the GPU) and the restriction to the few so called 'shader languages' for implementation.

[11] How, cf.
[12] cf. IMP, p.3 12-16.

1.2.4.3 FPGA Based

A Field Programmable Gate Array (FPGA) is a type of integrated circuit (IC) that allows to program its internal circuits. With a structure programmed it behaves like regular integrated circuits (ICs) and therefore can be used for data-processing.
Advantage is the reaction time, especially when used directly connected with Camera Link to the image source[13].
The relatively complex approach at programming and the high entry cost due to highly specialized circuit boards and programs are the disadvantages.

1.2.5 Interfaces

Machine vision as well as all universal industrial control systems require a series of interfaces to allow the different components to communicate with each other.
It is possible to distinguish between two major categories that build upon each other:

1.2.5.1 Data Transfer

The basic of communication is data transfer between devices. In the field of machine vision, three major systems are currently established. They are compared in the table 1.4[14][15].

NAME	USB 3.0	GigE	Camera Link	FireWire
Max Speed (data)	3.2 Gbit/s	1 Gbit/s	2.38 Gbit/s	0.6Gbit/s
Network/Direct	Direct	Network	Direct	Direct
Max Cable Length	3 m	100 m	10 m	4.5 m
Power Delivery	4.5 W	15.4 W	None	45 W

Table 1.4: Interfaces comparison

The formerly common standard Firewire is declining in prevalence[16].

[13] cf. Tre, Inline vs. co-processing, para. 2.
[14] cf. FLI, Table 1.
[15] cf. com, cpt. 1.3, para. 2.
[16] cf. FLI, Table 1.

1.2.5.2 ROS

Robot Operating System (ROS) is a State of the Art framework for developing applications involving robots. The particular focus is on operation in an industrial system. The developers state in quote 1.2.5.2:

> It [ROS] is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

Quote 3: [About ROS Fou, para. 1]

The basic interface and communication concept in ROS is displayed in diagram 1.1.

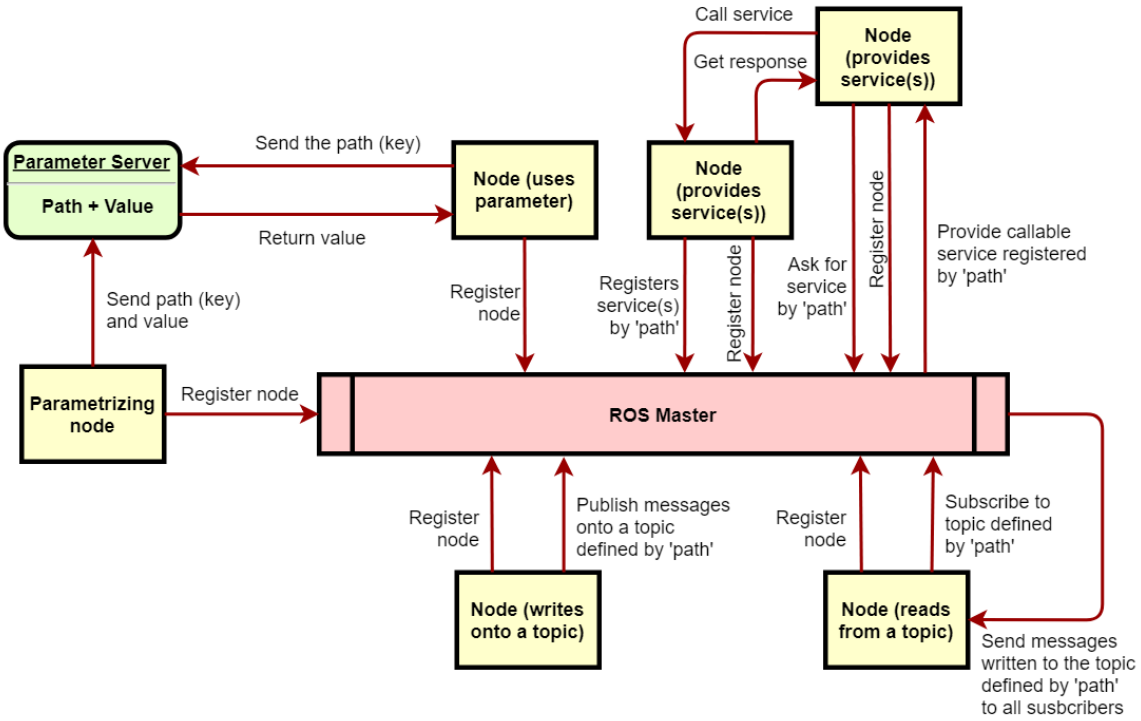

Figure 1.1: ROS Interface and Communication concept

A system using ROS contains the "ROS Master" that is provided as part of the framework and one or more named "nodes" that are developed for the desired system. After launching, every node has to register itself with its name to the ROS Master to be part of the framework.
The framework provides three major ways of communicating between registered nodes:

Writing and reading parameters to/from the "Parameter Server"
The parameter server is a node that is part of the framework and holds key-value pairs. The keys are strings and must follow a name convention[17] to comply with the

[17] cf. All, cpt. 1.1.

ROS guidelines. This way they form a path that contains the information which node has set up the parameter and what purpose it has.

Providing and calling structured services
Nodes can register services using a similar path-name like on the parameter service. A different node that knows the desired service-name can request a call to the service using the ROS Master. If specified, the service can return information to the caller.

Publishing and subscribing to topics Topics are similar to radio broadcasting. The ROS Master maintains so called "topics" that form a list of messages. Nodes (senders) can add to this list by "publishing" onto the topic that is defined by a path-like name. Other nodes (listeners) that have "subscribed" to this path will be notified about the new message. This is similar to the MQTT protocol[18].

A more in-depth description of service and topic communication can be found in chapter 3 Implementation.

"ROS is designed with distributed computing in mind."[19] Therefore communication with the methods described above is configurable to work with nodes being on different physical machines connected via ethernet in a local network.

Aside of the communication framework, ROS consists of a system involving so called "frames". They represent local coordinate systems that are related to each other forming a graph with parent/child relationship between every element. For example an industrial robot has a frame for each of its joints; directed from the robot base to the tool at the end of the robot. The ROS framework provides functionality to transform coordinates and orientations based on one frame, to any other frame as long as both frames are within the same graph. Further information about the usage of frames can be found in subsection 3.2.5 Applying Calibration (Transforming)

1.2.6 Typical Camera Installations

Industrial installations utilizing machine vision have a large number of details and variants due to the equally large number of problems they can solve. Due to the target use-cases of the company promoting this thesis, its focus lies on cooperation of machine vision with industrial robots. In this fields, the following two major installation concepts have been determined.

[18] cf. Tea, MQTT Publish / Subscribe.
[19] cf. Deb, Overview.

1.2.6.1 Static Camera

For this approach, the camera is fixed on a frame near the roboter. Usually in its robot-cell. The robot arm can move freely, pick, hold and place objects in the observation area of the camera as seen in the image 1.2.
This installation is used when the entire area where machine vision is required is a plane that is smaller than a suitable camera's view range. Common use cases are the verification of single products and the localization of objects on a fixed plate. Furthermore, it is preferred for installations that require robots with a payload that is not much bigger than a camera's weight.

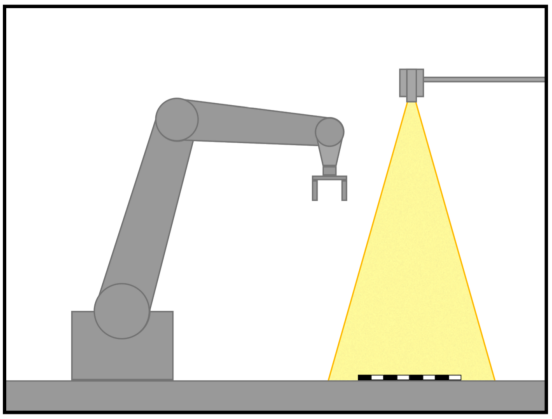

Figure 1.2: Basic Setup With a Static Camera[20]

1.2.6.2 Dynamic Camera

A setup with a dynamic camera means that the camera is attached as a whole to a movable part of the robot like the flange. In that case the camera is in a fixed position relative to the operation tool of the robot as seen in the image 1.3.

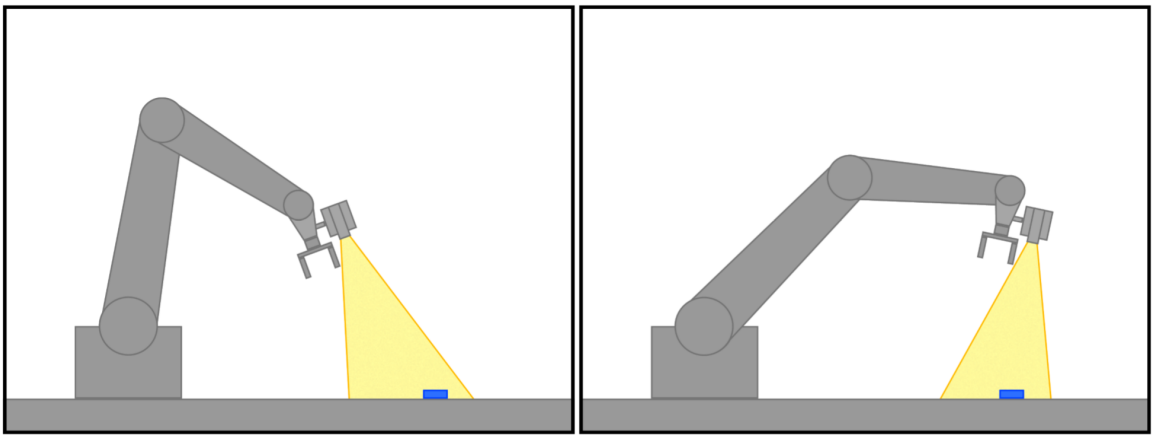

Figure 1.3: Basic Setup With a Dynamic Camera[21]

With the camera having the same agility as the robot, this construction is used when many different or large observation areas have been defined or if the target of observation is moving constantly. Examples for usage consist of close-up inspection of multiple products one after another and machine vision applications on a constantly moving conveyor belt.

1.2.7 Processes

Aside from hardware and communication concepts, machine vision requires certain processes to fulfill their part in an industrial system. Those processes can be categorized in three sections that are performed one after another and rely on data achieved by the preceding processes.

1.2.7.1 Calibration

The first procedure is a calibration of the machine vision system. The goal of this step is to form a relation between the information every component can perceive or provide. Additionally it can be needed to relate to parameters from the physical environment.
The scope of this thesis resumes to the context of industrial robots and the "extrinsic calibration" this field requires. Therefore it excludes purely optical corrections of images like distortion, chromatic and axial aberration as well as astigmatism[22]. The determination and therefore mitigation of those are referred to as "intrinsic calibration". Instead it is assumed that provided input image data has been prepared accordingly or the predicted error caused by said optical imperfections is within an acceptable margin.
Extrinsic calibration in a machine vision system that includes an industrial robot or other physical manipulator like conveyor belts, provides the possibility to transform locations and measurements performed on the image data of the camera, to the physical reality the robots interact with.
The two basic methods of installation, static- and dynamic-camera, require different procedures of calibration. Details about the difference are examined within the design and implementation chapters (see 3.2 and 3.2.5.2).

1.2.7.2 User-Applications

Aside from calibration, machine vision procedures require further information to fulfill their purpose and provide data to other parts of the installation. The systems have an user-interactable interface for this purpose. Four approaches that can partially be used together have been encountered during the research on machine vision systems from major manufacturers.

Browser-Based interface
Systems that are connected via GigE and support a web-based interface provide an interactive website that can be accessed through a common browser (e.g. Google Chrome, Mozilla Firefox). A system using this approach is the SmartCamera BVS002C from Balluff shown in this screenshot 1.4.

[22] cf. DG17, p.1, para. 2.

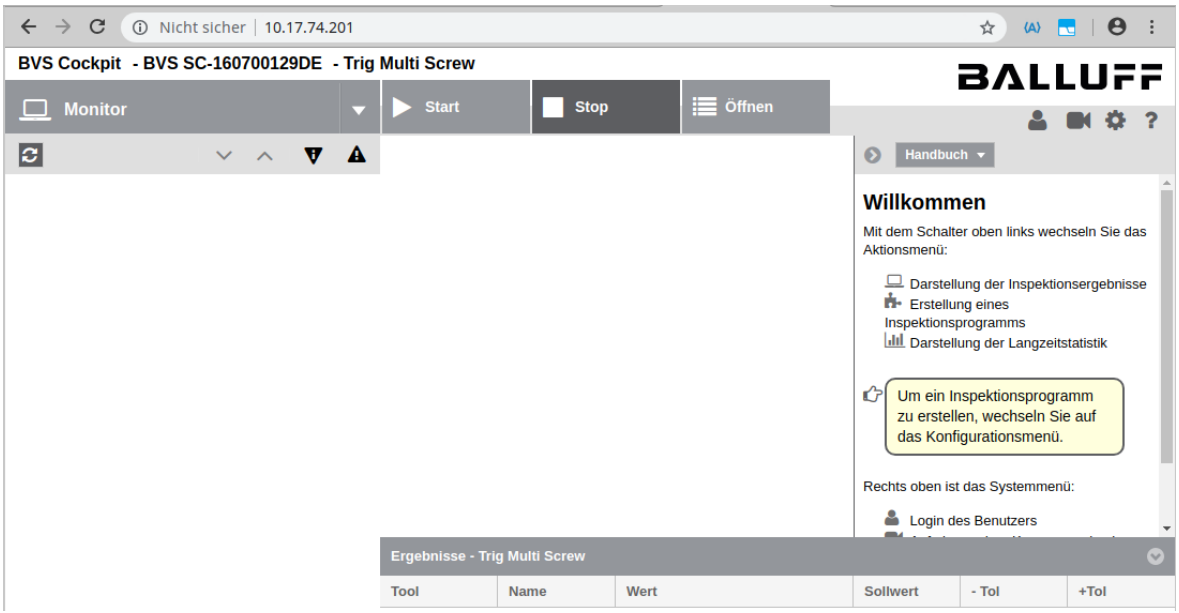

Figure 1.4: BVS002C interface called "Cockpit"

Software interface

A software needs to be installed on a target machine (or comes pre-installed if the hosting computer is provided by the manufacturer of the vision system).

Advanced wizards

Advanced wizards lead the user through the procedures of providing the machine vision system its purpose in the current installation. They consist of a tree-structure of steps the user can move through using the interface. An example is LumiTrax by KEYENCE[23]. It proposes the user multiple variants of possible results (for example for applied filters) and enables to chose the one appearing to be optically the most fitting.

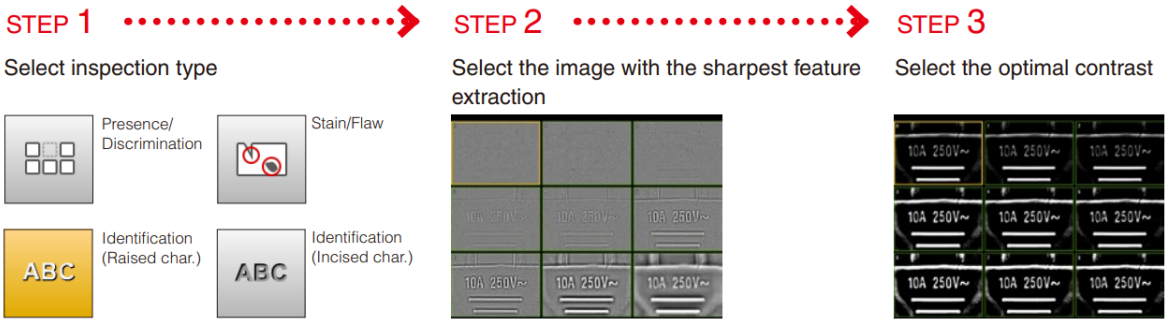

Figure 1.5: LumiTrax Steps[24]

[23]cf. Key, whole.

Drag and drop programming
This approach gives the user the possibility to design the machine vision procedure by selecting tasks and arranging them in a desired order with drag and drop gestures. Those tasks, also called 'blocks' can take and provide data values to each other to construct a procedural program. A system using this approach is the SmartCamera BVS002C from Balluff.

1.2.7.3 Teaching

Within the applications described above, certain blocks of information have to be provided to the machine vision system to adapt to the current usage. This procedure is often referred to as "teaching". One representative example which can be extended to other types of applications is the recognition and localization of objects. Those objects need to be taught to the machine vision system to provide the data needed to recognize the images.
As a State of Art example serves the teaching procedure of the Balluff BVS002C SmartCamera. While its overall application system is based on a drag and drop concept described above, the teaching mechanisms use a wizard consisting of several steps:

1. Chose a name for the new object to teach. This name can be later transmitted at runtime to the separate system that utilizes the localization.
2. Take an image of the plain background seen by the camera.
3. Place the object to teach below the camera in the base orientation that will later be recognized as "no-rotation".
4. Edit the proposed mask (1.6) where the localization algorithm will look for recognizable contours.
5. Finalize and select the area in the camera-view where to expect the object

An alternative approach at teaching objects for localization is to provide many samples of possible objects and enable the machine vision system to determine automatically what areas are of interest. This is often used in systems involving neuronal networks and so called "deep-learning"[25].

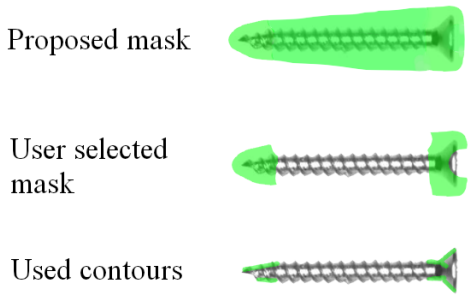

Figure 1.6: Mask for Object Recognition - Desired: Recognizing only the tip and the head of the screw

[25] cf. Cogb, whole.

1.3 Industrial Programming

Aside from machine vision, other components of industrial installations require user-applications and teaching as well to achieve that they work together. A field that is highlighted in this context within this thesis are industrial robots.

1.3.1 Manufacturer Specific

Equivalently to manufacturers from the machine vision field, ones of industrial robots provide their in-house teaching system and user-applications. The most often encountered approach are handheld-devices tied to the robot that allow to pre-program series of tasks for the robot that can later be activated at runtime. Samples for those so called "controllers" can be seen in the images 1.7.

Figure 1.7: Handhelds (HMI/Controller) from Universal Robots (left)[26] and Kuka (right)[27]

1.3.2 Alternative: drag & bot

> "drag & bot is an intuitive web-based software for easy setup, programming and operation of industrial robots."

Quote 4: [drag & bot db, top]

As the quote 1.3.2 describes, the software developed by the drag & bot GmbH which is a spin-off of Fraunhofer IPA aims to provide an universal environment to program industrial installations that consist of robots from different manufacturers and accessories like machine vision systems. Its usage should not require in-depth knowledge about programming or robots.

1.3.2.1 Concept

Drag&bot consists of three parts visualized in diagram 1.8.

1. The front-end that allows the setup of robots and other components, direct control of the robots and development of programs through a drag-and-drop interface.
2. The robot-system that performs the commands from the front-end and executes the programs.
3. The back-end that manages programs and accounts.

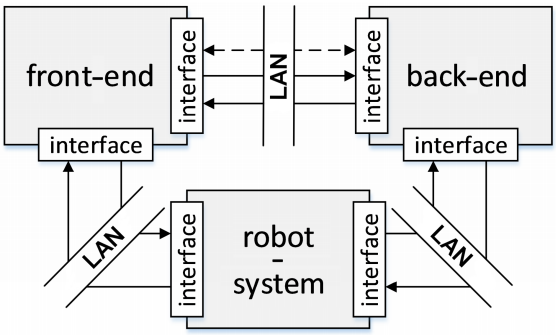

Figure 1.8: System Architecture: drag & bot[28]

Every part is connected to the others through ROS-messages and -topics or websockets. This allows them to run on separate linux machines connected by an ethernet network. Support of Microsoft Windows is currently being tested.

1.3.2.2 Components (drivers)

The "robot-system" part consists of so called "components" which can be understood as "drivers". Those are written in C++ or Python and form ROS nodes to allow communication with the rest of drag and bot. They access the hardware directly and expose the desired functionality as ROS-services. Additionally a .yaml file provides information about what configuration parameters are required for the component. The front-end utilizes this file to provide the end-user an interface for those parameters.

1.3.2.3 Function Blocks

Function blocks are the elements the programs in drag & bot are composed of. Every function block consists of input and output variables as well as Python code that processes those variables. The front-end provides a code editor for the end-users to write their own function blocks in Python and therefore expand the functionality if desired.

The image 1.9 shows a possible program that is made out of function blocks. Its purpose is to use a SmartCamera to localize certain objects, pick them up with the gripper of a robot and, depending on whether they are labeled as "Triangle" or not, release them at two different positions. This is an example for using an industrial system to sort randomly placed objects.

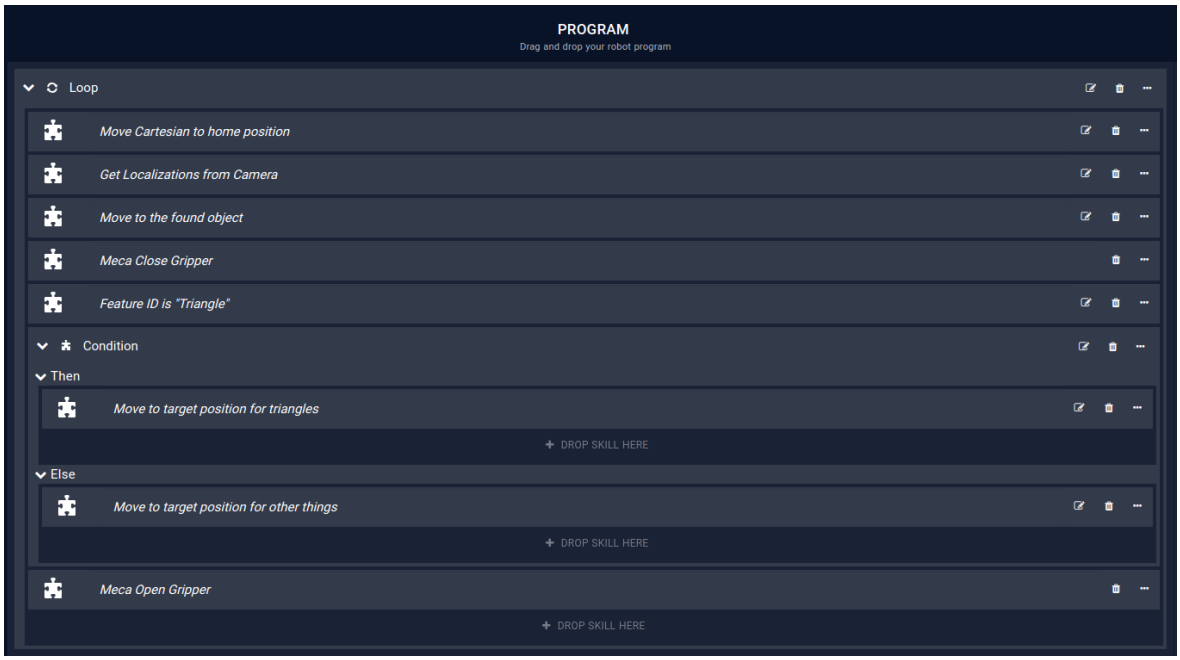

Figure 1.9: Sorting Program in drag & bot

1.4 Motivation: Machine Vision for drag & bot GmbH

The drag & bot GmbH aims to make common use-cases of the robotics industry easily accessible and at comparatively low cost for the end-user as well as for developers.

1.4.1 Problem Definition

Implementations of machine vision in drag & bot are specific for a single industrial installation. A new machine vision system requires re-implementing the support of the new hardware and features into the whole user-application, including the front-end and the robot-system. This results in a significant demand of resources for the company and slow delivery to the customer.

1.4.2 Goals and Research Question

The goal for the drag & bot GmbH is a solution that fulfills the following major characteristics:

Reusable
The solution should be implementable for multiple components that each access hardware from different manufacturers. Major tested examples during the work on this thesis were:

- The Balluff BVS002C SmartCamera
- The calibration system with the SmartCamera as well as a simple 2D camera

Extensible
Additional procedures and functionality should be adaptable with a minimum of re-implementation of components and minimal adaption of the front-end.

Independent
To avoid cross-dependencies to external companies, the solution should be rely on in-house developments and do not add relations to closed-source projects.

Maintainable and Documented
To achieve usability on long term, the solution has to be clearly structured and designed in a maintainable way as well as documented thoroughly.

Furthermore, the implementation is desired to fulfill standard qualities of State of the Art software like code-readability and performance.
Through the analysis of the State of Art implementations above advertised by major manufacturers and from direct experience provided by the drag & bot team it has been concluded that manufacturers impose a high dependency on support for usage of their hardware. Eventually implementations that use that hardware in the way instructed by the manufacturer do not allow reusability when switching between manufacturers and partially even between products of the same manufacturer. The result is that universal interfaces are required to allow the efficient implementation and reuse of concepts within drag & bot and to fulfill the goal of independence. Furthermore, a complete abstraction is needed to achieve that the end-user can work with subsubsection 1.3.2.3 Function Blocks independently of the manufacturer of the used machine vision hardware.

Consequently, the research question this thesis is based on, comprises the goals described above and asks how an interface can be designed, that allows the implementation of a solution satisfying the needs of the drag & bot GmbH: easy of use and integration, reusability, adaptability and extensibility.

Chapter 2

Development of Interfaces and Concepts

This chapter describes and substantiates the design of the machine vision interface that has been developed for drag & bot and gives an overview of the program structure. Further implementation details are provided in chapter 3 Implementation.

2.1 SmartCamera Driver

As described earlier in subsection 1.2.3 Smart Cameras, this type of cameras internally processes images and provides output data that can be used directly by the surrounding system through a driver ("component" in drag & bot). However different SmartCameras have different manufacturer-specific interfaces and required procedures. Consequently to fulfill the goal of reusability, the driver component requires an universal interface in ROS that forms the connection to the other components in drag & bot.
The process of designing consisted of evaluating what minimum functionality is required to make smart cameras universally usable and enable the user of the driver to utilize all functionality drag & bot supports, including ROS' topic system (see Figure 1.2.5.2 ROS).

Two basic concepts were developed and evaluated: A stateless and stateful approach. The major difference is that the order of accessing the different features of the interface is predefined and that order needs to be followed when using the interface. The stateless variant on the other hand allows a usage of features independently of what accesses have performed before or will perform afterwards. Effectively stateless means that no component maintains information about what calls have been made beforehand.
Benefits of the stateless approach is a smaller interface with less usage complexity compared to the stateful approach. Downside is a higher amount of information

required to be transmitted for every interface access and, depending on the smart camera, less predictable and more varying execution time. The reason is that the implementation has to keep track of the smart camera's internal state.
Opposed to that, the stateful approach increases complexity of the interface with features to handle state. Therefore execution duration is constant (except for network delay) and the implementation is closer to the smart camera's internal implementation.
Because execution-time-predictability is significant in commonly real-time controlled robot environments and because the drawback of higher complexity has been evaluated as insignificant, the stateful approach is being used.

In the concrete planning phase, the interface elements were named and described. At this point it has been discovered that the current sample of a smart camera (the Balluff BVS002C) provides functionality that is not supported by every alternative smart camera the interface is required to be compatible with to fulfill the goal of "reusability". A compromise between constricting the feature set and universality has been found, that involves defining interface elements and parameters that may be optional, thus their usage depending on the actual implementation.
The following three mandatory interface elements have been determined to be required for managing the smart camera:

Setup (establish connection)
Requires connectivity data like IP address or USB port and triggers connecting to the smart camera. A provided "camera name" is used to differ between multiple cameras connected through the same driver to enable use in parallel. Additionally a keyword provides what type of data the camera outputs (for more information, see chapter 3 Implementation). Furthermore, this feature takes a ROS-topic where the detailed status of the camera is published and updated at runtime. If the smart camera offers procedures that can run on their own (for example automatically localize objects in a loop), it is possible to activate this at this step.

Enable Publishing
This element enables that all data responses from the camera (whether triggered by other features, or automatically sent due to an internal loop) are published onto a provided topic.

Stop/Unload (disconnect)
Stops all activity of the camera, including stopping any running loops and removes it from the internal list of available cameras. Afterwards, the setup feature is accessible again.

One optional interface element has been defined:

Change Application/Task
For smart cameras that have an internal programming structure, this interface element can be used to chose the currently active application or task.

Smart cameras provide a large number of features. Examples have been given in subsection 1.2.3 Smart Cameras. For the interface that means that those features require to be accessible. A theoretical solution was a single interface element that triggers the current application/task or a specific feature chosen by a key-word, no matter what output data type it provides.
However, due to the constrictions of the framework used in drag & bot, ROS, the implementation of such an interface turned out to be not possible. A reasoning can be found in section 3.1 BVS002C Driver. Instead, with the research in mind that the feature-sets of different SmartCameras overlap significantly, the interface itself has been designed in an extensible way.
As a result, every feature induces an additional interface element.

Trigger Application/Task/Feature of TYPE
For smart cameras that have an internal programming structure, this interface element performs the currently selected application/task and optionally allows to switch temporarily to a new application/task. For smart cameras that provide features only, those are called and identified by a key-word. In both cases, the type of the response has to be transmitted within the interface access.
If required, the actual implementations of those interface elements may contain additional input data that is transmitted to the smart camera.

2.1.1 ROS services

Eventually the concept feature elements have been turned into specifications for ROS services. This communication construct described earlier in Figure 1.2.5.2 ROS, is defined by a name, a series of input parameters and a series of output parameters. Actual implementation additionally requires a data type. The following graphical overviews contain a description for every parameter.

The three mandatory services are described inf figures 2.1, 2.2 and 2.3.

SmartcameraSetup: SERVICE - INPUT

Param Name	Param Description
camera_name	Name for the camera. Used for info messages and to access multiple cameras if desired. Has to be unique!
camera_model_name	Camera model. For example 'BVS002C'
port	Network port for the camera or USB port
con_address	If camera is connected by network: IP address to access the camera (otherwise ignored)
application_id	The ID of the application/algorithm on the camera to use when starting
interpret_as	Keyword determining the way of interpreting the response from the application. For example: "Localizations" or "50ByteStrings"
topic_status	Topic to publish the status arrays on. Empty string to ignore

SmartcameraSetup: SERVICE - OUTPUT

Param Name	Param Description
error	Error message. Empty if everything went well. Otherwise it should be displayed to the user and offer retry

Figure 2.1: Interface ROS Service: SmartcameraSetup

EnablePublishing: SERVICE - INPUT

Param Name	Param Description
camera_name	Name of the camera like provided on setup
topic_for_outputs	Topic to publish the resulting data on. Ensure that the topic type matches the datatype!
min_delay	Minimum time between two responses from the camera.
queue_size	Maximum messages to enqueue onto the topic at once

EnablePublishing: SERVICE - OUTPUT

Param Name	Param Description
error	Error message. Empty if everything went well. Otherwise it should be displayed to the user and offer retry

Figure 2.2: Interface ROS Service: EnablePublishing

SmartcameraStop: SERVICE - INPUT

Param Name	Param Description
camera_name	Name of the camera like provided on setup. This way multiple cameras can be accessed

SmartcameraStop: SERVICE - OUTPUT

Param Name	Param Description
---	NO OUTPUT but the camera has been closed and disconnected

Figure 2.3: Interface ROS Service: SmartcameraStop

The optional service is described by figure 2.4.

ChangeApplication: SERVICE - INPUT

Param Name	Param Description
camera_name	Name of the camera like provided on setup. This way multiple cameras can be accessed
application_id	ID number of the application to switch to

ChangeApplication: SERVICE - OUTPUT

Param Name	Param Description
error	Error message. Empty if everything went well. Otherwise it should be displayed to the user and offer retry

Figure 2.4: Interface ROS Service: ChangeApplication

Two examples for feature-specific services are visualized in figure 2.5 and 2.6.

Trigger50ByteString: SERVICE - INPUT

Param Name	Param Description
camera_name	Name of the camera like provided on setup. This way multiple cameras can be accessed
application_id	The ID of the application or task on the camera to switch to and trigger. If it is not the current application or task, it will switch back afterwards!
timeout	If the application on the camera does not work as expected, waiting will be aborted after this timeout (in s)

Trigger50ByteString: SERVICE - OUTPUT

Param Name	Param Description
error	Error message. Empty if everything went well. Otherwise a failure reaction should be executed
strings	An array of strings received from the camera application. Each up to 50 characters long

Figure 2.5: Interface ROS Service: Trigger50ByteString

This interface element is mainly for testing new smart cameras as it expects the camera to reply with 50 byte long strings.

TriggerLocalization: SERVICE - INPUT	
Param Name	**Param Description**
camera_name	Name of the camera like provided on setup. This way multiple cameras can be accessed
application_id	The ID of the application or task on the camera to switch to and trigger. If it is not the current application or task, it will switch back afterwards!
timeout	If the application on the camera does not work as expected, waiting will be aborted after this timeout (in s)
TriggerLocalization: SERVICE - OUTPUT	
Param Name	**Param Description**
error	Error message. Empty if everything went well. Otherwise a failure reaction should be executed
localizations	List of a message type that represents and contains the information about a localized object

Figure 2.6: Interface ROS Service: Localization

The TriggerLocalization interface element is for practical use as it returns the coordinates of objects localized by the camera. More details, especially involving the internal design of the "localizations" parameter's datatype can be found in section 3.1 BVS002C Driver.

2.2 Calibration Interface

The camera calibration has the purpose to transform coordinates from the plane of a two-dimensional image to the three-dimensional coordinate system used for example by an industrial robot. To achieve the required data for that, a user-controlled procedure needs to be performed and be accessible from a visual interface (in the case of this thesis, the front-end of drag & bot).
As a result, the required interface consists of two parts: Interface to the front-end and interface to perform the mentioned transformation. Both cases have to fulfill the goal of reusability and extensibility.
The process of developing both interface segments was similar to the procedure for the SmartCamera driver but in a preceding step, a conceptual design for the software had to be found. The reason was the goal of extensibility.

As described earlier in subsection 1.2.6 Typical Camera Installations, there are two concepts of calibrations (static and dynamic) and several possible implementation

variants that are further described in section 3.2 Universal Camera Calibration. Therefore, the software structure is required to be adaptable and extensible in many details while maintaining a single interface. A point of significance is the information provided to the user and the user-input, because those differ between different types of calibration.

One possible solution involved the design of a front-end for every supported calibration type and a key-word based selection of the currently demanded design. Alternative is a remote-design concept that enables the calibration procedure to transmit to the front-end what information to show to the user and what user-input to expect.
The former process concept has the benefit of a straightforward implementation, but the drawback of requiring code modifications in two places (and in the case of drag & bot in two programming languages) whenever extending or changing the camera calibration system.
The remote-design concept requires a comparatively complex interface design but the required implementation of the front-end is restricted to the usage of the interface, independently of which calibration methods are implemented.

Eventually the decision has been made for the remote-design due to its extensibility and the prospect to re-use the concept itself in different parts of drag & bot, if it turns out to be useful in practice. Furthermore, reverting to a traditional, separate front-end is relatively easy as described in section 3.2 Universal Camera Calibration. The core of a remote-design is a sequence-concept that determines the basic design of the software.

2.2.1 Sequence Concept

The goal of the sequence concept was to design a machanism and an interface that allows components of the robot-system (in this case the calibration-system) to communicate with the user through the front-end without tying the exact implementation of the robot-system to the front-end. To achieve this goal, it has been decided that the whole communication shall be split into a series of steps that together form a "sequence".

For every step the sequence consists of, the robot-system waits for a communication message from the front-end that contains data, the user has provided for the last step (or none, if there was no last step). This data is processed for the last step and then all needed information for the current step is passed to the front-end which displays it to the user. This data consists of two sections: Firstly, Informative text to provide information to the user. Secondly, requests to the user to provide input

that is required for the step.
The process diagram 2.7 visualizes the basic procedure with respect to the three active sections: The robot-system, the front-end and the frame.

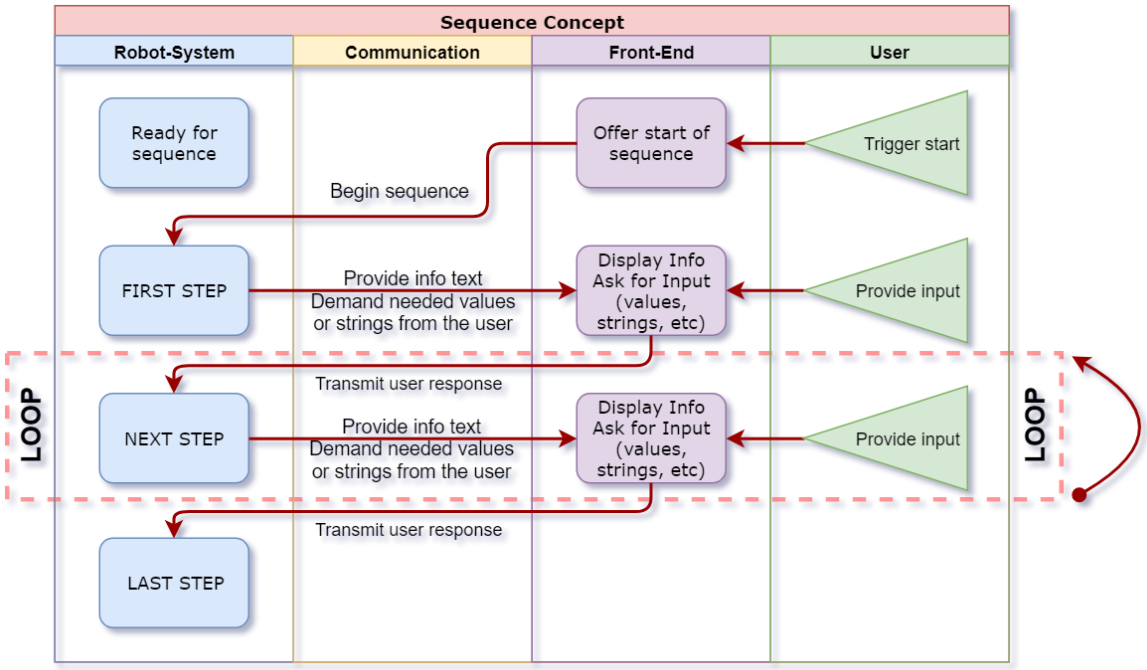

Figure 2.7: Sequence Concept Overview

Key element of the design is the fact that no information origins from the front-end. Instead it only passes information on, that has been provided by the robot-system or the frame. This results in an independence between the structure of the procedure within the robot-system and the front-end.

2.2.2 ROS services

Similar to the development of the SmartCamera driver interface, a sufficing minimum number of interface elements were designed and later on tansferred into ROS-services that consist of parameter lists.

The following interface elements and ROS services have been designed for the robot-system component "Camera Calibration":

Start Calibration

Accessed on command from the user to begin calibration. Exchanges ROS-topics and default data that is required for calibration. For in-depth details see the chapter 3 Implementation.

StartCalibration: SERVICE - INPUT

Param Name	Param Description
topic_image1	Image topic that is also published to the frontend (Empty string to use the node parameter)
topic_image2	Image topic that contains source images from a simple camera or 'SMARTCAMERA' for using the smartcam
checkerboard_ squaresize_m	Default value for the size of a square on the checkerboard if calibrating with simple camera
tf_frame_robot_tcp	TF frame the contains the position of the robots tool tip the user calibrates with.
tf_frame_robot_base	TF frame of the fixed base for the calibration. If given an empty string, the robot base will be read and used
fixed_camera	Whether camera is fixed on the robot cell 'True' or 'False' if attached to flange. Note this is a string. If providing an empty string, the calibration will ask the user.

StartCalibration: SERVICE - OUTPUT

Param Name	Param Description
---	NO OUTPUT but calibration is ready for the first step

Figure 2.8: Interface ROS Service: StartCalibration

Perform Current and Get Next Step

Takes current user-input data and directly provides information to display to the user by the front-end for the next step. Due to implementation-specific details, the information is separated into multiple parameter by type: Strings (texts), values (numbers) and options (multiple-choice selections).

PerformCurrentAndGetNextStep:SERVICE - INPUT

Param Name	Param Description
repeat_last_step	If true, the last interaction step will repeat
abort	If true, the interaction process aborts and can restart
value_inputs [ ]	List of numeric user-responses from the last step
text_inputs [ ]	List of text user-responses from the last step
option_inputs [ ]	List of text user-responses from options provided in the last step

PerformCurrentAndGetNextStep: SERVICE - OUTPUT

Param Name	Param Description
error	Error message. Empty if everything went well. Otherwise it should be displayed to the user and offer retry
header_text	Text to display as the title of the current step
description_text	Info text to display on the frontend. It describes what to do in this step
button_text	Text for the button that calls the next step
descriptions_for_ needed_values [ ]	For every element in this array there should be a numeric field with this text as a label/description
default_ needed_values [ ]	Array of default values for every needed numeric value to apply to the numeric fields
descriptions_for_ needed_strings [ ]	For every element in this array there should be a text-field with this text as a label/description
default_ needed_strings [ ]	Array of default texts for every needed string to apply to the text fields
descriptions_for_ needed_options [ ]	For every element in this array there should be a drop-down menu with this text as a label/description
default_ needed_options [ ]	Array of options to provide in every drop down menu. Each string has the pattern: DEFAULT|OPT1|OPT2 ...

Figure 2.9: Interface ROS Service: PerformCurrentAndGetNextStep

Export Calibration

Writes the calibration into a file at a given location. Used by the front-end to enable the user to chose where to save a calibration for backup-purposes.

ExportCalibration: SERVICE - INPUT	
Param Name	**Param Description**
calibration_name	Calibration name given when having performed the calibration.
target_path	New path and file
ExportCalibration: SERVICE - OUTPUT	
Param Name	**Param Description**
error	Error message. Empty if everything went well. Otherwise it should be displayed to the user and offer retry

Figure 2.10: Interface ROS Service: ExportCalibration

Load Calibration

Loads an existing calibration that has been exported before.

LoadCalibration: SERVICE - INPUT	
Param Name	**Param Description**
calibration_name_ or_path	If name: Loading from the current DnB runtime. If path containing '/': Load file relative to user directory
LoadCalibration: SERVICE - OUTPUT	
Param Name	**Param Description**
error	Error message. Empty if everything went well. Otherwise it should be displayed to the user and offer retry

Figure 2.11: Interface ROS Service: LoadCalibration

Transform Coordinates

This interface element provides a direct way to translate two-dimensional coordinates on an image taken by the associated, calibrated camera into three-dimensional coordinates in the given reference frame (on frames see Figure 1.2.5.2 ROS).

TransformPixelPose: SERVICE - INPUT

Param Name	Param Description
header	Header to identify the service call
seq	Increasing identity number
time	Time stamp
frame_id	Frame the pose is based on
x	Coordinate (for example on an image) in horizontal direction from left to right
y	Coordinate (for example on an image) in vertical direction from bottom to top
theta	The "roll" rotation of the orthogonal point
tf_frame_reference	Frame the resulting response pose should be based on.

TransformPixelPose: SERVICE - OUTPUT

Param Name	Param Description
pose	Pose containing that transposed coordinates
header	Header to identify the service call
seq	Increasing identity number
stamp	Time stamp
frame_id	Frame the pose is based on
position	Coordinates on the corresponding axis
x	X axis position
y	Y axis position
z	Z axis position
orientation	Threedimensional orientation as a quaternion
x	X component
y	Y component
z	Z component
w	W component

Figure 2.12: Interface ROS Service: TransformPixelPose

2.3 Machine Vision with drag & bot

To fulfill a task within an industrial installation that is controlled by a drag & bot system, the two machine vision components "SmartCamera Driver" and "Camera Calibration" have to work together as well as involve the drag & bot mechanism subsubsection 1.3.2.3 Function Blocks that eventually use their data. It is possible to differ between two phases that are described and visualized in the following sections.

2.3.1 Preparation Phase

The first phase involves all preparations required to achieve an evaluable setup. The base of this process flow is given by the design of drag & bot and involves preparation of the external components (robot and (smart-)camera, including teaching), addition of the so called "components" through the front-end, parametrization of all components and design of a program using function blocks. Eventually the calibration procedure described in section 3.2 Universal Camera Calibration for the camera has to be performed.
The flowchart 2.13 visualizes the steps.

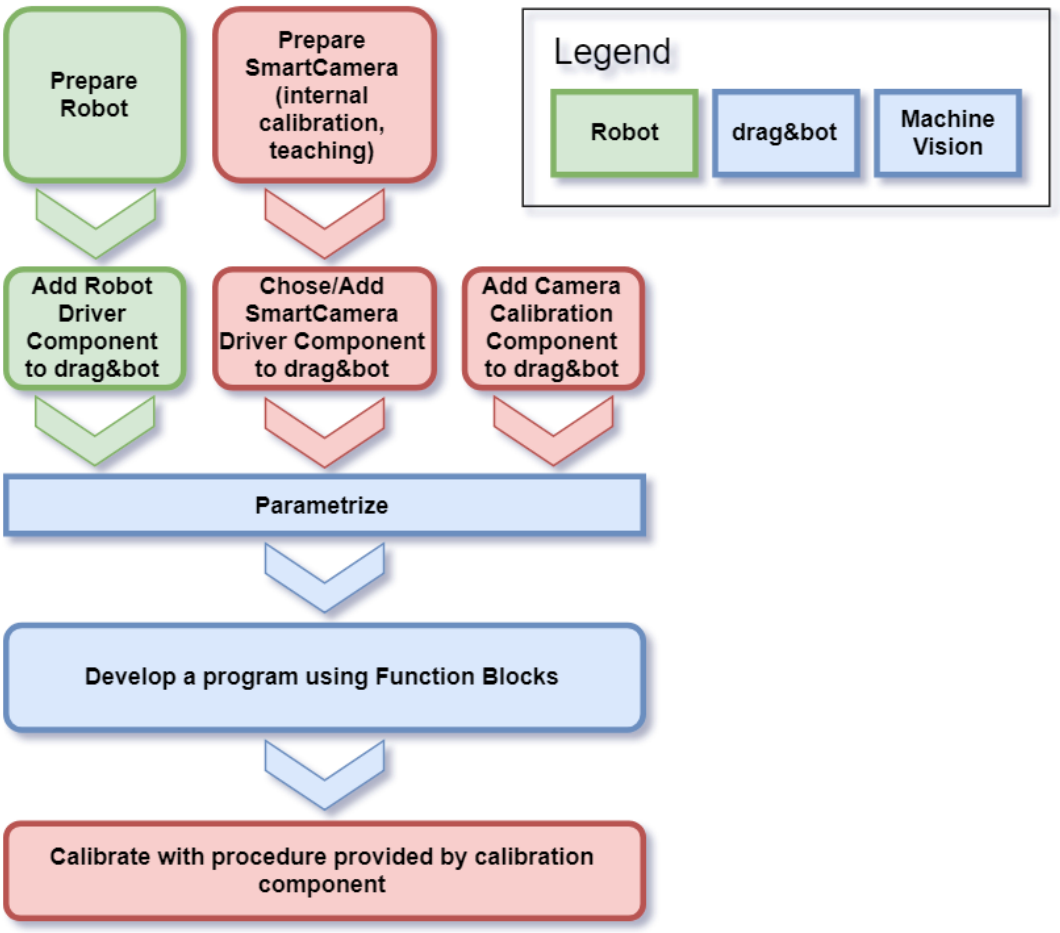

Figure 2.13: Flowchart of the preparation procedure

2.3.2 Performing Phase

The performing phase basically consists of the execution of function blocks. An executor component of the drag & bot environment calls the Python code of every block one after another and transmits input- and output-parameters between function blocks in the way the programmer has designed it in the preparation phase. To use machine vision with function blocks, two data-flow designs have been conceptualized that require each a different design of the function block and implementation of the extensible SmartCamera driver. They have been evaluated on the common example of localizing pre-taught objects with a SmartCamera and moving the robot to the resulting, calibrated coordinates.
In the first variant (A) shown in 2.14, the SmartCamera driver provides raw uncalibrated coordinates of the localization result to the function block. Then the calibration is applied before the output parameters for other function blocks are prepared.
Alternatively the data flow 2.15 (B) can be implemented to use the camera calibration from within the SmartCamera driver and directly output calibrated three-dimensional coordinates and orientation to the function block.

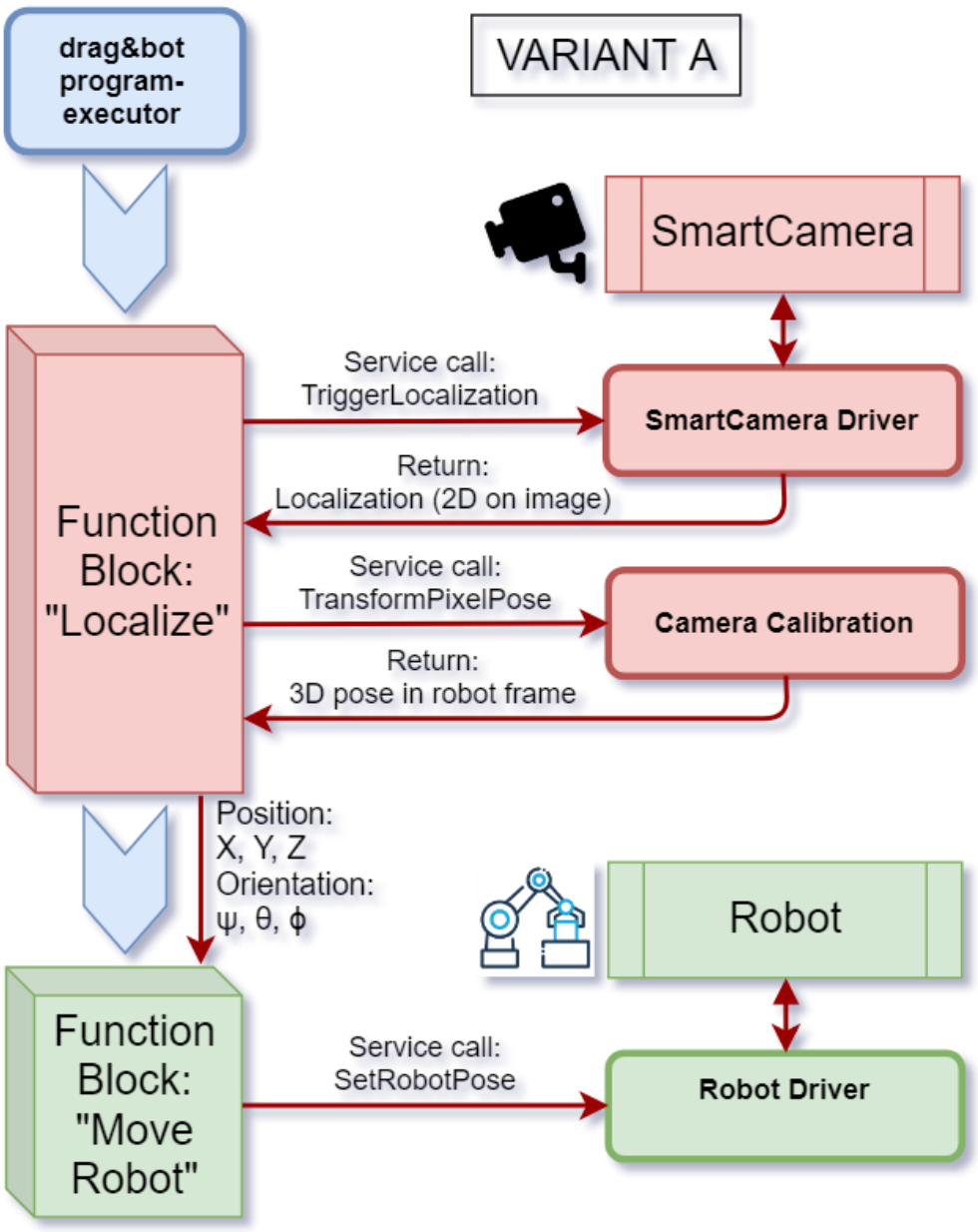

Figure 2.14: Localization Program (Dataflow variant A)

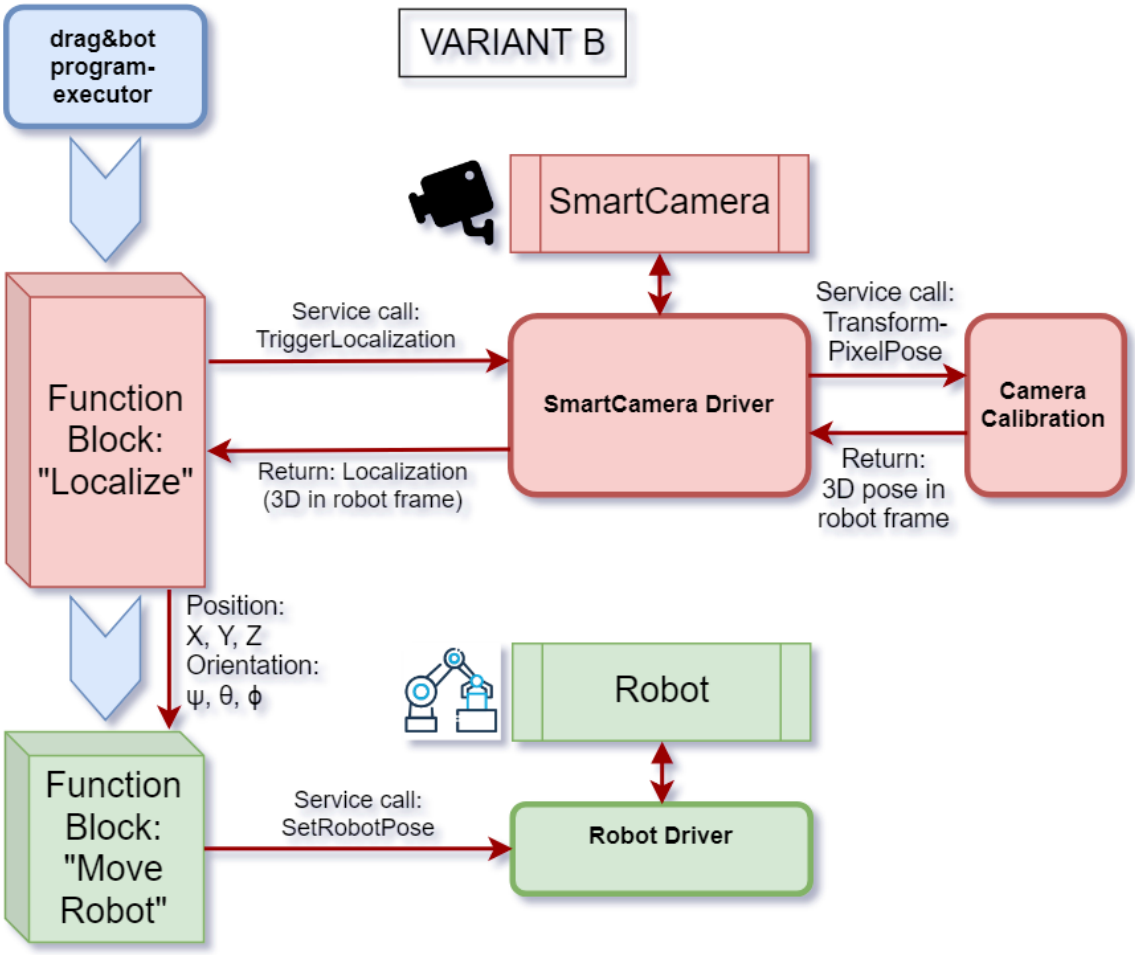

Figure 2.15: Localization Program (Dataflow variant B)

The advantage of variant A is the higher versatility of the dataflow and a more informative behavior in the cause of incorrect preparations or parametrization because the function block can handle error cases of the SmartCamera driver separately from the usage of the calibration. In comparison to this, variant B results in significantly simpler function block code and therefore is easier usable for an external customer who desires to write their own, modified function block for a machine vision installation.

At the time of this thesis, active demonstrations at the drag & bot GmbH use variant A, however minor modifications would be required to adapt to variant B. For more information, see section 3.2 Universal Camera Calibration.

Chapter 3

Implementation

With the interface elements conceptualized, the next step was to implement the software for drag & bot and use the interfaces. To achieve compatibility with drag & bot, all program code had to be written in the programming language Python and encapsulated by so called "ROS-nodes". Nodes are programs either written in Python of C++ that are handled automatically by the ROS framework. They require a name to be identified and can communicate through services and topics with each other as described earlier in subsubsection 1.2.5.2 ROS.

3.1 BVS002C SmartCamera Extensible Driver

As described earlier in section 1.2 State of Art, the camera is the base of a machine vision system. The drag & bot GmbH desires to use a SmartCamera from the local manufacturer Balluff, the BVS002C. Therefore this was the first candidate to implement the SmartCamera Driver for.
That SmartCamera is based around so called "applications". Those are internal programs that can be programmed through an integrated website that is provided by the camera itself to any browser that has LAN-access to the camera. Those applications describe what the camera will do in a fixed order. For example, first take an image, then apply a filter to said image, localize certain objects on it that have been taught along with the programming procedure and then send the results value after value in a programmer-defined pattern. Several datatypes like REAL32, STRING and BOOL are supported. Applications can also retrieve input values from the user, however this functionality has not been implemented so far because encountered use-cases did not require this and therefore were not taken into account for the design of the interface.

An application programmed can be either "triggered" from a user computer that utilizes the camera, or it can run on its own in a loop, performing the program and sending the data over and over automatically. The camera can contain multiple

of those applications, each with an unique ID number, and while the last-loaded application will be re-loaded when starting the SmartCamera, it is possible to switch the applications at runtime. Wrapping all this functionality into a class was the first step towards implementing the driver.

3.1.1 Communication with the BVS002C

The SmartCamera can communicate with the computer through web sockets via ethernet. Commands consisting of a series of bytes are sent to the camera and responses are received to fulfill a statefull protocol. The complete protocol of the BVS002C can be found in its handbook which is also accessible via the integrated browser interface of the camera. Therefore this chapter does not go in depth about every aspect of the communication, but rather give an overview with focus on key-aspects that had been taken into account when implementing the interface.
All commands sent to the camera require to consist of the following three or four byte-blocks:
1. The "magic number"; the 4 byte long unsigned integer value "1347638850" encoded as a little-endian ("42565350" in hex format). It is used to verify the correct transmission of the current message.
2. Another 4 byte long unsigned integer containing how many bytes the remaining user-data will contain.
3. A 4 byte long unsigned integer containing a message-id from a given list supported by the camera.
4. Last block is optional and can have any byte length to carry further data that is required for some commands.

Since the protocol is stateful, a connection message is the first that needs to be sent to the camera. However due to possible issues with formerly established connections, the implementation first sends the closing-message that is without effect if no connection exists (see subsection 3.1.4 Challenges.
The message-id for establishing a connection is 01_{HEX}. Therefore, since the byte length of the user case is just the 4 bytes of the id, the resulting message consists of the following hex values (the blocks are separated by white space): 42565350 04 01 That command responds to the user with a message beginning with the same pattern (magic number, user-data-length, the calling command-id) but adds another unsigned integer that describes the protocol version.
The list of supported message-ids encompasses but is not limited to "*switch application*", "*start application*", "*trigger application*", "*reset*", "*status*" and "*response-container*". The last two are only found in received messages because they are not immediate responses to commands sent by the user.

The basic communication with the SmartCamera has been implemented in the class **"balluff_smartcam_connector"**. Its constructor takes the parameters required to access a certain camera (the ip address and the port) and it abstracts the detailed communication with the camera. Therefore it provides class methods with no or few parameters. For example `connect_software()` and `set_application(application_id)`. Furthermore, it ensures that responses from the camera are handled as well. This required especial focus to ensure that the protocol of the camera is followed accurately.

That protocol consists of messages that can be sent actively from the camera at any point in time, without a directly preceding command from the user. Therefore it has been required to split up the flow of the program into sending tasks and receiving tasks. To ensure that no message is lost, incoming messages from the camera are placed onto a queue which is then handled one after another in a separate thread. The last driver aspect the **"balluff_smartcam_connector"** handles, is publishing of status- and optionally, application-results onto the ROS-topics provided through parametrization. To generate application-results, no matter whether for publishing, or for returning a trigger method, the **"application_response_interpreter"** further described in subsection 3.1.2 Extensibility and Responses is used. Also see the UML structure of the driver in subsection 3.1.3 Node and Overview.

3.1.2 Extensibility and Responses

The goals of the company (subsection 1.4.2 Goals and Research Question include extensibility of the interface and therefore of the implementation as well. For the SmartCamera driver that means that it can be extended to support new applications. More accurately that means new application-responses (the REAL32s, STRINGs, BOOLs etc. programmed to transmit results) can be handled, because the driver has no direct contact with the content of an application but only with its output- and optionally its input-data.

To achieve an adequately comfortable and reliable way of extending the driver, the handling of application-responses has been encapsulated in a separate class: **"application_response_interpreter"**. Its purpose is to take lists of bytes that are application-responses from the camera and convert them into predefined data-structures (ROS-messages) that can be published to topics or directly used by function blocks through the trigger-services.

As described earlier in subsection 2.1.1 ROS services, the interface utilizes parameters named like "interpret_as" which take a keyword containing the information how the application shall be interpreted. This keyword is passed to the interpreter method along with the list of bytes from the camera. Depending on the keyword,

a corresponding method made specifically for interpreting the given list of bytes is called. A series of `get_TYPE(first_byte)` methods have been implemented to convert the byte list to variables in Python. "TYPE" in the function name is the datatype like "bool" or "real32" and "first__byte" is the index of the first byte within the list that belongs to the demanded datatype. The method `get_string(first_byte, length)` differs because it requires a length parameter containing how many bytes shall be interpreted as a string.
Those methods are used to extract an instance of the specified result-datatype. If the list of bytes is longer than required for one instance of the result-datatype, the method starts interpreting anew with the remaining bytes. This way, multiple result-instances of the same type can be transmitted from the camera at once.

To visualize how this interpretation concept works, two diagrams have been designed. 3.1 shows how multiple 50 characters long strings are programmed to be sent by the camera (left column), are interpreted by calls to `get_string(first_byte, length)` (centered column) and eventually are turned into the built-in type string-array (right column); the result-datatype that's returned from the method.
For comparison, look at the interface element Figure 2.1.1 ROS services which is used to pass this result to a function block. Note that the BVS002C SmartCamera automatically fills up shorter strings to the given byte-length with zeros which are automatically cut-off by the `get_string()` method.

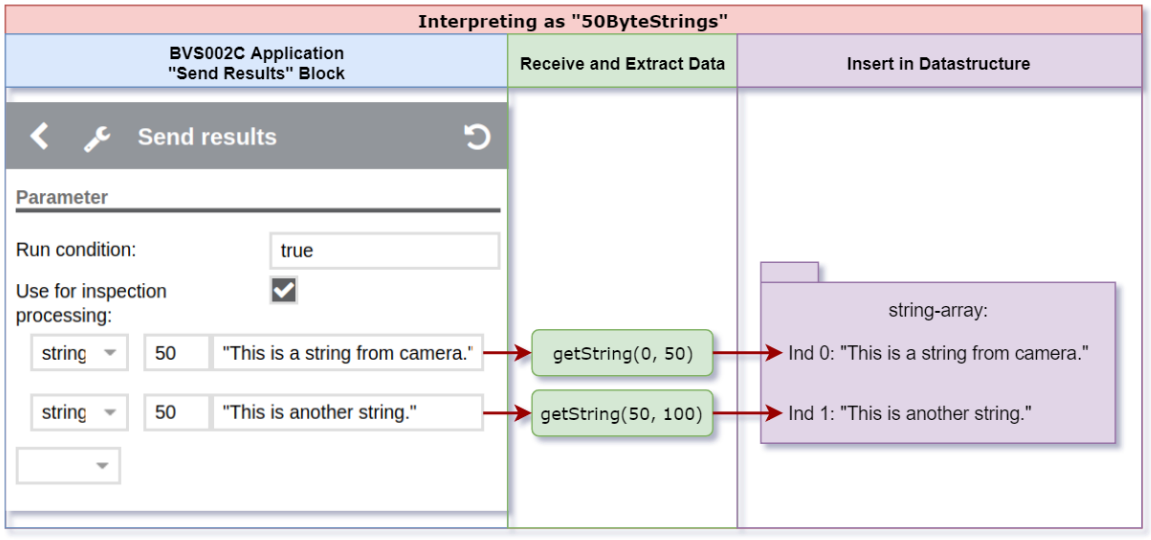

Figure 3.1: Output-Interpretation for an application that provides 50 byte long "strings"

This second diagram 3.2 shows the interpretation of a more complex camera-application that sends all values needed for a ROS-message named "Localization" which is the result-datatype in this case. Again an example of usage for this output is the corresponding trigger- interface element (see Figure 2.1.1 ROS services).

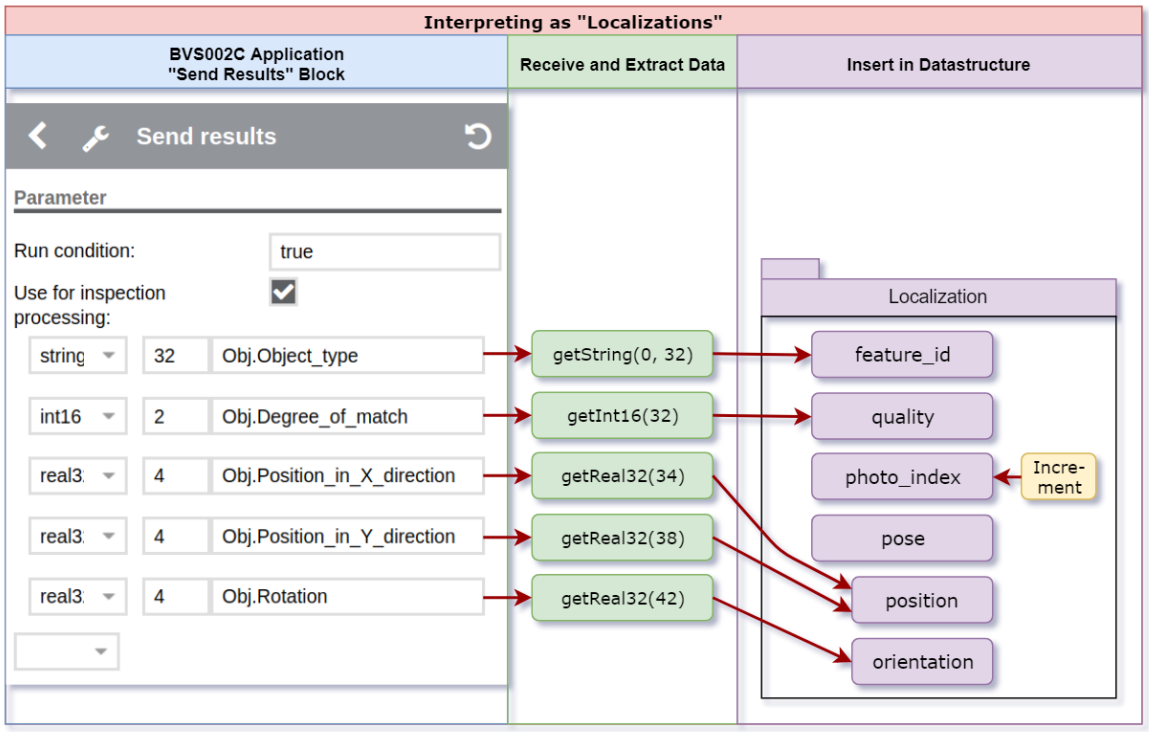

Figure 3.2: Output-Interpretation for an application that provides "localizations"

Features of the SmartCamera like this, that do not result in a datatype already available in Python (like the string-array) require a specially designed type, defined as a so called ROS-message. The diagram in 3.3 describes the ROS-message named "Localization".

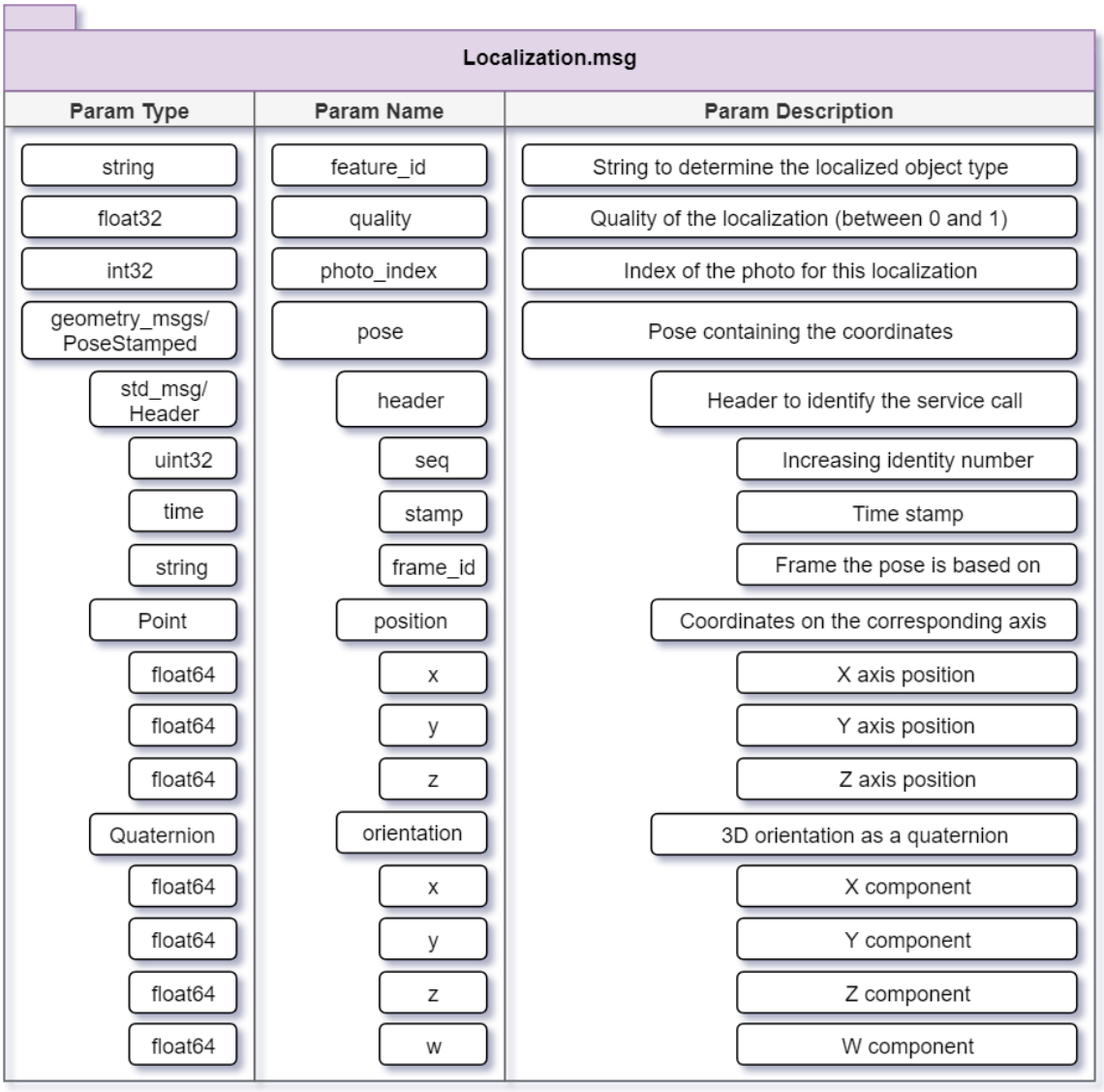

Figure 3.3: The ROS-message "Localization"

Note that with the goal of re-usability in mind, this datatype is equally used for two-dimensional localizations directly from the camera as well as three-dimensional results available after a calibration has been applied to the localization. See 3.2 for further information. This datatype is based on a preexisting design work at drag & bot GmbH[1].

[1] cf. Vis18, p.76 "detection.msg".

3.1.3 Node and Overview

Eventually a class to create and utilize instances of "balluff_smartcam_connector" has been implemented: "balluff_smartcam_server". It contains the required method calls for the ROS-node and provides all the services developed at design phase (subsection 2.1.1 ROS services). An overview of the whole driver structure can be seen in the UML diagram 3.4. As a demonstration it shows how two different cameras are connected through the same driver.

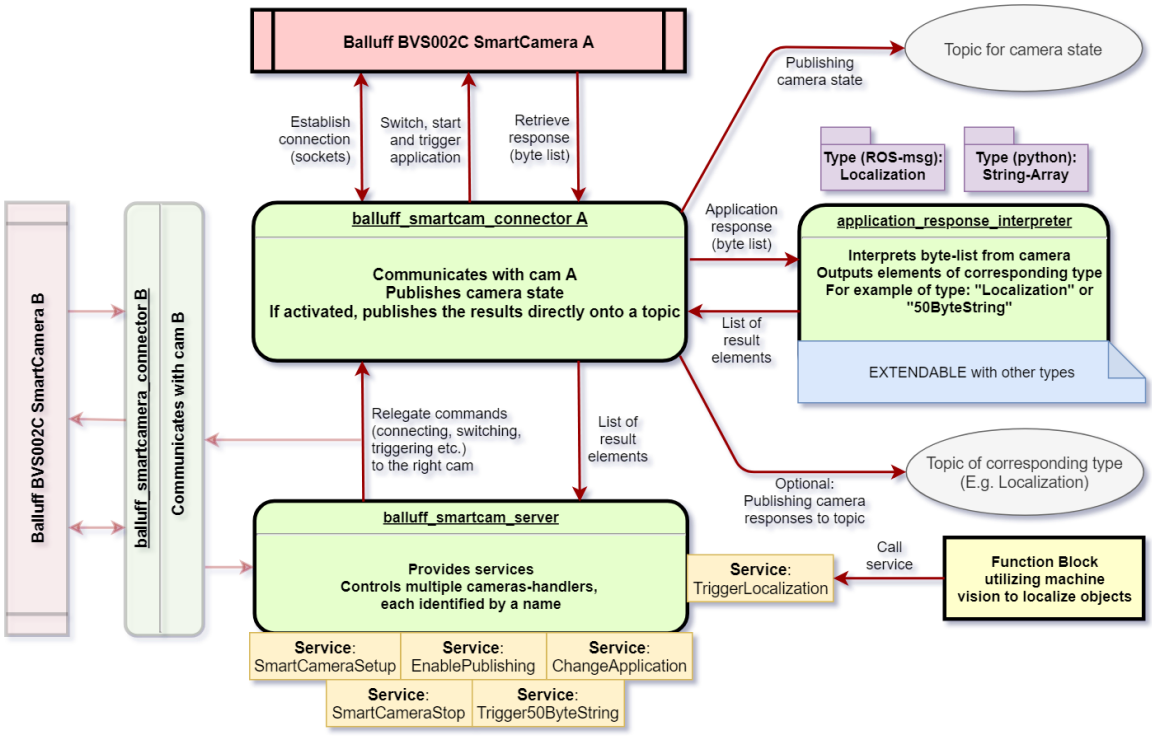

Figure 3.4: Overview of the Extensible SmartCamera Driver

3.1.4 Challenges

The development of the driver possessed a number of challenges that required compromises or rethinking of parts of the implementation. This section gives a quick overview including the taken counter-measures that might need to be taken into account for implementing the interface for other SmartCameras.

Camera state not always known or unpredictable

Due to the nature of the drag & dot, the driver can be shut down independently of the camera. Furthermore, the camera can be taken off power any time and it is possible to change the camera's state through the integrated website independently of the driver. The result is unpredictable state at start-up of the driver and partially at runtime as well (partially because some state changes are transmitted through the status messages, but not all). Furthermore, despite the documentation of the

BVS002C SmartCamera claiming that the re-establishing of a new connection overwrites the current state, it can happen that the first connection attempt fails in (so far) incomprehensible ways or results in unexpected messages.
The two machanisms that has been implemented as a counter measure are a procedure to send disconnection messages always at start up before re-connecting. Additionally, the disconnection and connection code is wrapped in a loop that restarts in case the camera sent an unexpected message response. Unfortunately the result is a significantly increased driver-start-up time that depends on the state of the camera on driver-start-up.

User-input/application depending wait times
The internal protocol of the SmartCamera requires specific wait times between two commands. For example between switching to a new application and starting that application. Unfortunately the time needed to load an application depends significantly on its content. For example an application that has several objects taught to recognize will take longer to load than one that did not require teaching. The solution has been to check for the "BUSY" status provided by the camera to wait as long as needed.

Possible bugs
Since the BVS002C is still partially in development, bugs and inconsistencies with the manual have been encountered. A recent software update improved the situation noticeably.

3.2 Universal Camera Calibration

Approaching the implementation of the calibration consisted of three major topics:

Sequence concept
Determining an efficient way to implement the sequence system that has been designed as part of the interface.

Required input
Determining what data is required to be able to transform the two-dimensional points to the three-dimensional robot space and what tasks the user needs to do.

Calculations
Determining the required calculation for transforming coordinates.

This chapter leads through those aspects.

3.2.1 Sequence Concept

To implement the sequence concept as described in 2.2.1 the programmatical ability of Python has been used, that allows to pass methods just like values to other methods. The method stored in a parameter can then be executed later with a call. Therefore the series of separate steps required for the calibration process have first been defined in individual methods without parameters.

This way it was possible to develop a class named frontend_step_system with a method `expect_step(function, info_text, input_containers)`. The first parameter is the method required for the current step of the sequence. "info_text" contains the header-, message- and button-text to show to the user. Last, the "input_containers" is a list of a container data structure that describes information is needed from the user. For example it asks for a binary value (boolean) whether the camera is fixed relative to the robot's base or attached to its flange.

When called, this method immediately starts to wait for a call to the interface service Figure 2.2.2 ROS services. Once this occurs, it utilizes the *TYPE_input* parameters to fulfill the "input_containers" of the **last** step (the currently passed "input_containers" are not used at this point yet). Then it executes the **last** provided method (the "function" parameter of the last call), unless it has been "None". If executing it, raised an error, the call to the interface service is immediately responded by providing the error parameter. If the execution passed successfully, the error is an empty string and a more complex response is formed that transforms the newly given "input_containers" into the separate *needed_TYPE* arrays. Once this response is sent, the call to `expect_step(function, info_text, input_containers)` finally returns.

Simplified, this construction results in the possibility to program the calibration process in the pattern shown within the pseudocode 3.5. It consists of just four steps of which only three can execute. The user decides by input which last step will execute.

```
# Create an instance of the sequence handler. It provides the ROS-Service.
sequence = frontend_step_system()

# Input container for a binary question
camFixed = InputContainer("Is the camera fixed?", BOOLEAN)
# Show the question to the user and Wait for the first user-response
# No function required yet
sequence.expect_step(None, "Answer the question.", [ camFixed ])

# Another question for the user (this time demanding a string)
camName = InputContainer("What is the camera name to calibrate with?", STRING)
# Take the response of the user for the camFixed-question and ask for camName.
sequence.expect_step(None, "Answer the second question.", [ camName ])

# Now, camFixed has its user-response set! That can be accessed with get()

if (camFixed.get() == True) # If the user said the camera is fixed
    # Ask to perform fixed-calibration
    sequence.expect_step(calibrate_fixed_camera, "Press OK, to calibrate now.", [])
else:
    # Otherwise ask to perform fixed-calibration
    sequence.expect_step(calibrate_dynamic_camera, "Press OK, to calibrate now.", [])


# Functions

# Will be executed if the user answered the first question with "True"
def calibrate_fixed_camera():
    # "camName", the response to the second question is available now anywhere
    print "Calibrating FIXED camera named: " + camName.get()

# Will be executed if the user answered the first question with "True"
def calibrate_dynamic_camera():
    # "camName", the response to the second question is available now anywhere
    print "Calibrating DYNAMIC camera named: " + camName.get()

```

Figure 3.5: Sequence System Code (highlighted)

The real, implemented system also offers the option to abort the current sequence by calling the service with the corresponding ”abort” flag set to True. In a similar way it is possible to repeat the last step if it failed or the user is not satisfied with a result. The actual calibration functions and the steps following the given sample above, are implemented in the class *general_extrinsic_calibration*. If an implementation without the sequence system would be desired, it is possible to expose the functions of this class as ROS-services and handle their calls in the front-end.

3.2.2 Calibration Method and Required Data

Calibration in this context means to find a way to transform from the coordinate system of the camera, to the coordinate system of the robot. A procedure to achieve this relation has been implemented based on the following concept: Let the user (the calibrating person) provide positional data in the robot-coordinate-system for which corresponding positional data within the camera-coordinate-system is known. A way to retrieve coordinates from the robot that is already implemented in drag & bot is to use the position of the tool center point (TCP). For example for a gripper

that's mounted on a robot, the TCP is usually at the center of its gripping-area. drag & bot provides comfortable ways of controlling the whole robot by moving this TCP through the front-end. The orientation of the TCP is available as a quaternion as well.

Two-dimensonal cameras that are currently in focus of the drag & bot GmbH have two-dimensional values as output values that need to be transformed. For this reason it can be deduced that all three-dimensional points the calibration will transform to, are on exactly one pane in the robot's space. Mathematically a way to express a pane is one point in space and a normal vector that is exactly orthogonal to it. Furthermore, the transforming needs a scale factor because the two-dimensional values for the camera are usually in pixels (unless the SmartCamera scales internally) and the robot's system works with meters. Together with the point and the vector, that is the whole information required from the user.

3.2.3 Origin Point and Scale

The first two parts of required information are the arbitrary point on the three-dimensional pane and the scale factor from camera-coordinates to robot-coordinates. To achieve this information the approach when using a SmartCamera differs from the usage of raw images. To fulfill the goal of re-usability, both cases have been implemented and are described in the following subsections.

3.2.3.1 With SmartCamera

As shown earlier as an example in 3.1, SmartCameras like the one used by drag & bot supports the localization of objects or images. Therefore a so called "calibration sheet" has been designed specifically for SmartCamera. The sheet seen in 3.6 has been designed specifically for the Balluff BVS002C contains the recognizable origin-point pattern at the top left as well as a circle-pattern that is required for internal, robot-independent calibration developed by Balluff. Both have to be on the same sheet, because internal calibration determines how the two-dimensional orientation of the origin-point is determined. Furthermore, the two orthogonal rays 'A' and 'B' are on the calibration sheet. Those are required later for subsection 3.2.4 Normal Vector.

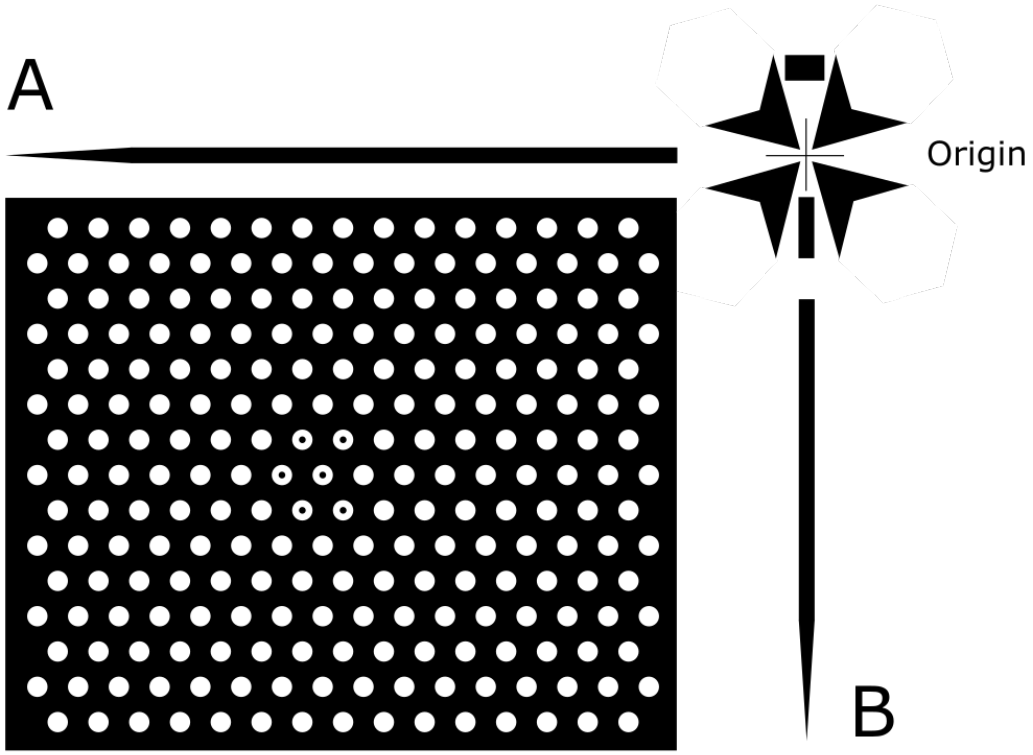

Figure 3.6: Calibration Sheet for BVS002C (circle-pattern by Balluff)

If prepared correctly, the SmartCamera can localize the origin to extract the position and orientation. Afterwards the user can move the TCP accurately to the origin point as well and thus achieve this point in the robot's coordinate-system as well. The orientation of the TCP is not required.
In the case of the SmartCamera, an explicit scaling factor is not required either because this type of camera provides coordinates in real dimensions (in meters) already. However internally, a scaling factor has been implemented by Balluff.

3.2.3.2 From Raw Image

Achieving origin point and scaling from raw two-dimensional images required a different approach. With development-efficiency in mind, a trivial implementation has been chosen that bases on localizing a checkerboard on the calibration sheet. An implementation that has been evaluated as sufficient is found in the vision library OpenCV[2] and is used for the implementation of calibration. However while it provides corner-points of the checkerboard as well as the scale-factor (if the real size of a checker square is provided), the orientation of the checkerboard is not known and therefore an unique origin-point cannot be determined.
To solve this problem, a dense pattern is located at a known distance relative to one outer corner of the checkerboard on the calibration sheet. The areas around

[2]cf. papers mentioned Ope, pp. l. 623–640.

all other corners remain white as seen in 3.7. This makes it possible to extract the average brightness value of each area (marked with blurred circles on the sheet) and compare them. The corner that had the darkest area is assumed to be the unique origin point and is thus known within the camera's coordinate system. By moving the TCP to this origin point on the calibration sheet, the location in robot-coordinates are determined as well.

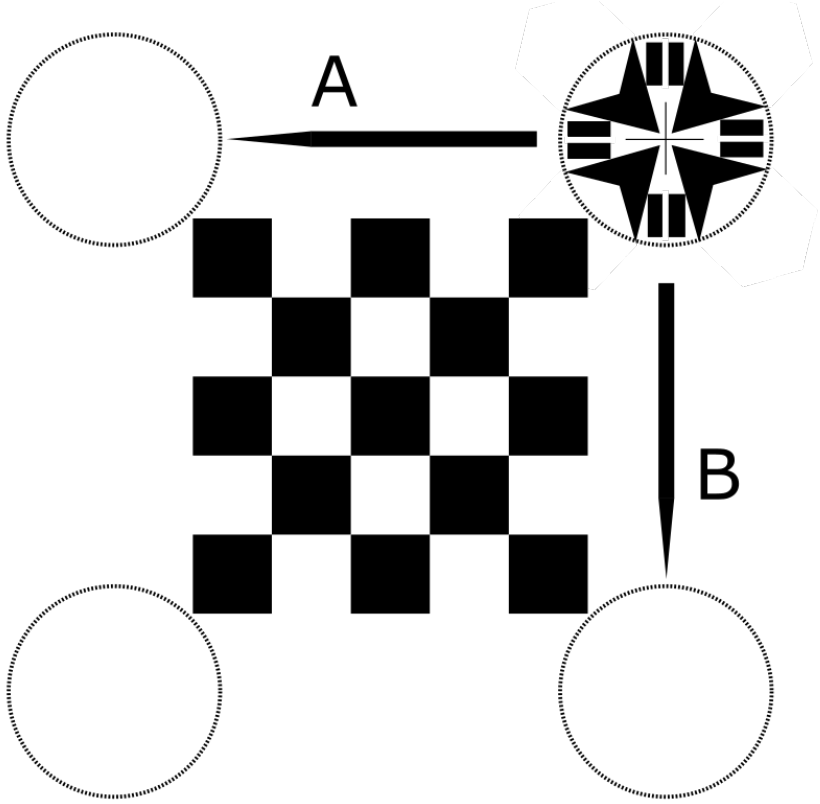

Figure 3.7: Calibration Sheet for Raw Images

Again the calibration sheet contains the two orthogonal rays 'A' and 'B' required for the next section.

3.2.4 Normal Vector

The last piece of information required to determine the three-dimensional pane is the normal vector. Technically the user could adjust the TCP's orientation correctly to be on axis with the camera and in the same angle as the origin-point. In practical tests it turned out that manually adjusting TCP in a desired way is not comfortable and inaccurate for real users. Therefore the normal of the desired pane is determined in a different way by performing two similar steps: The user has to move the TCP to a random position along the 'A' axis of the calibration sheet and then repeat the step with the 'B' axis. Those two points, subtracted from the coordinates of the origin point (which is at the intersection point of the two axes), allow to compute the normal by applying the dot-product on them.

3.2.5 Applying Calibration (Transforming)

Inspired by subsection 2.2.1 Sequence Concept, the actual procedure of applying the calibration to a new set of coordinates is working in separate steps. Again a feature of the Python programming language is used to allow that a list of objects (called "solvers" in this chapter) are passed to a loop (3.8). That loop calls the `apply_solver(input_pose)` method for each solver by beginning with the given, "non-transformed" pose and then using the output of the last solver as the input parameter for the next solver.

```
1  def transform_point_from_cam_to_robot(input_pose):
2
3      output_pose = input_pose
4
5      for calibration_solver in list_of_solvers:
6          # Apply every solver one after another
7          output_pose = calibration_solver.apply_solver(output_pose)
8
9      return output_pose
10
```

Figure 3.8: Method that Applies the List of Solvers

Each solver is implemented to do a certain part of the transformation between the camera's coordinate systema nd the robot's. This mechanism allows for clear separation of the tasks done by every "solver" and it fulfills the goal of extensibility because additional solvers can be added without altering internal interfaces. The section subsubsection 3.2.5.2 Dynamic Calibration describes such an implemented but untested addition.

3.2.5.1 Implemented Solvers

As of the time the development process for this thesis has finished, two solvers were finalized. Together those apply a static calibration. Static in this context means with a camera that is attached to the robot cell at a fixed position within the robot's coordinate system.

solver_camera_2D_position

This solver applies the 2D transformation of the camera only. That means first it adds the position of the origin-point in the camera's coordinate system to the new pose. Secondly it applies the rotation of the origin-point to the resulting point by applying the vector-rotation formula seen in 3.9.

```
out_x =  in_x * cosinus(angle) + in_y * sinus(angle)
out_y = -in_x * sinus(angle)   + in_y * cosinus(angle)
```

Figure 3.9: Rotate Vector by Angle

solver_real_world_position

The second solver applies the actual transition to the robot's three-dimensional coordinate system. From the information about the pane determined during calibration it has computed a quaternion that transforms a vector from the camera (z is assumed to be 0 because the camera's internal pane in two-dimensional) to the three-dimensional pane. The algorithm to achieve this has not been developed by the maker of this thesis but is an adapted variant of an online resource[3]. Because the solver_camera_2D_position has subtracted the origin-point's position and orientation, the solver_real_world_position can now pretend that the coordinates it has taken for input, are already a vector. Therefore the first step of applying this solver, is to transform the vector by the pre-computed quaternion.
This is performed by the calculations seen in 3.10 where the method `multiply_quaternions(q1_q2)` performs matrix multiplications on the two quaternions. `get_magnitude()` provides the length of the vector and `conjugate()` provides the conjugated version of a quaternion.

```
# Get the magnitude
mag = in_vec.get_magnitude()

# Form a pseudo-quaternion from the normalized vector and w = 0.0
quat2 = [
    x = in_vec.x/mag,
    y = in_vec.y/mag,
    z = in_vec.z/mag,
    w = 0.0 ]

# Apply the new vector to the transformation
transformed_quat = multiply_quaternion(quat1, quat2),

# Multiply the resulting quaternion with the conjugate
# of the original transformation
transformed_quat = multiply_quaternion(transformed_quat, q1.conjugate())

# Revert to a vector and scale by the original magnitude
out_vec = [
    x = transformed_quat.x*magnitude,
    y = transformed_quat.y*magnitude,
    z = transformed_quat.z*magnitude ]

```

Figure 3.10: Applying the Transformation Quaternion to a Vector of Input Coordinates

[3]cf. pen, p.3 12-16.

Afterwards the resulting vector is re-added to the three-dimensional coordinates of the origin-point of the pane. The result is the final, completely transformed position. The orientation for the output-pose is computed by forming a quaternion with a rotation axis that is equal to the normal of the pane and a rotation angle corresponding to the two-dimensional rotation of the input (see 3.11).

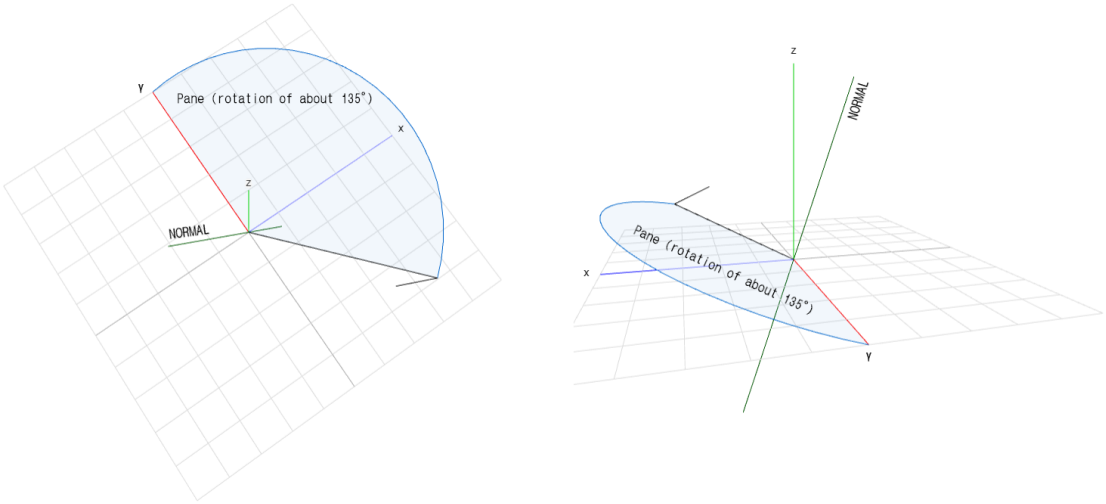

Figure 3.11: A quaternion that describes a slightly skewed pane and a rotation of 135°. Left: Seen from above. Right: 3D view.

3.2.5.2 Additional Solver for Dynamic Calibration

Dynamic calibration means that the transforming from camera-coordinate system to robot-coordinate system maintains correctness even for the case that the camera is not fixed (statically) within the robot-cell but is attached to the robot. Technically any part of the robot is a possible fixture as long as a pose (or ROS-frame) with fixed relation to the camera is retrievable.

To maintain the extensibility and make use of the effort that went towards reusability, the idea behind the current implementation of dynamic calibration is to have an additional solver that is applied before all other and "undoes" that additional transformation that exists because the camera moves with the TCP. Its transforming process subtracts the rotation and translation between the current pose and the pose used when taking the calibration-image. So in the end it allows that the other solvers can pretend to work further as if part of static calibration.

Unfortunately due to the temporary unavailability of a robot capable of carrying the SmartCamera, the soundness of this implementation could not be verified so far.

3.2.6 Class Overview

The figure 3.12 shows an overview of the implemented classes and their major purpose in the context described in the last sections.

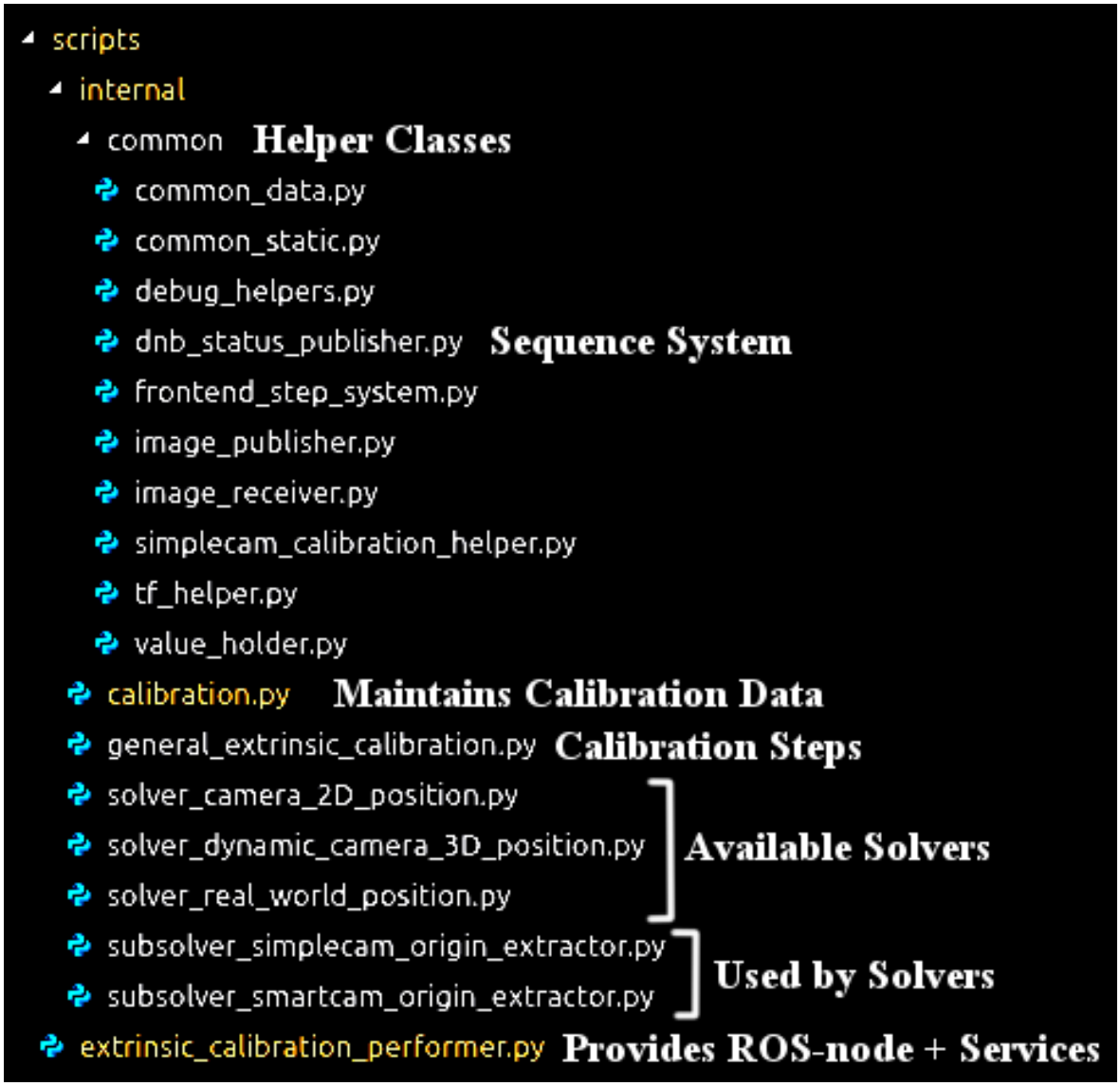

Figure 3.12: Classes Implemented for Calibration

3.2.7 Preview Output

During development of the subsection 3.1.2 Extensibility and Responses with the sample application that localizes objects, it appeared helpful to have a screen that shows where the SmartCamera had localized things. During implementation of the calibration, this preview-window has been extended to show how locations with and without calibration applied, relate to each other and also how the two coordinate systems relate. Because three-dimensional rendering would have exceeded the development resources for this thesis, the preview has been reduced to the two-dimensional appearance.

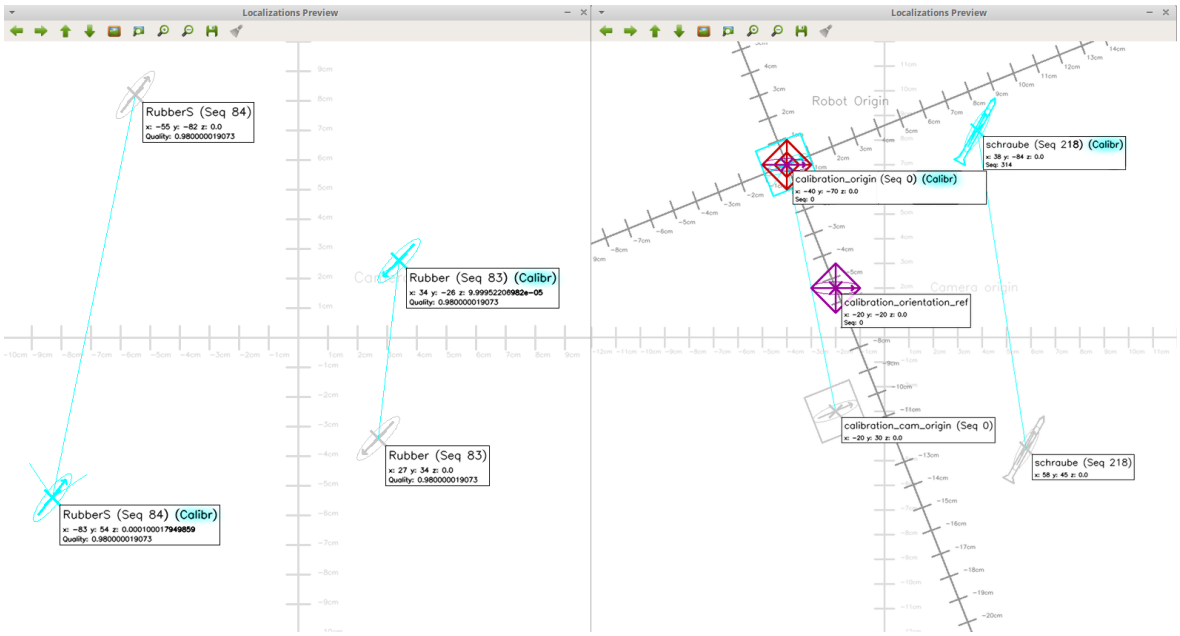

Figure 3.13: Preview Windows. Left: Two transformed objects. Right: Transformed objects with both coordinate systems visualized.

As it can be seen in 3.13, the preview window shows positions in the camera-coordinate-system (gray) as well as in the robot's system (blue). If the calibration process itself has been performed with the preview-window active, it also shows the localized origin-point and visualizes both coordinate systems (on the right side where two overlapping coordinate systems are shown).

3.2.8 Reference Implementation Front-End

During the implementation of the "sequence system", a reduced implementation for the front-end has been developed as well. The main purpose was testing but it also serves as a reference implementation for other team members in drag & bot GmbH. Unlike the real front-end that is based on AngularJS, this front-end (3.14) works with tKinter. It consists of a couple of buttons that control the SmartCamera driver and during calibration it provides messages and textfields for user input.

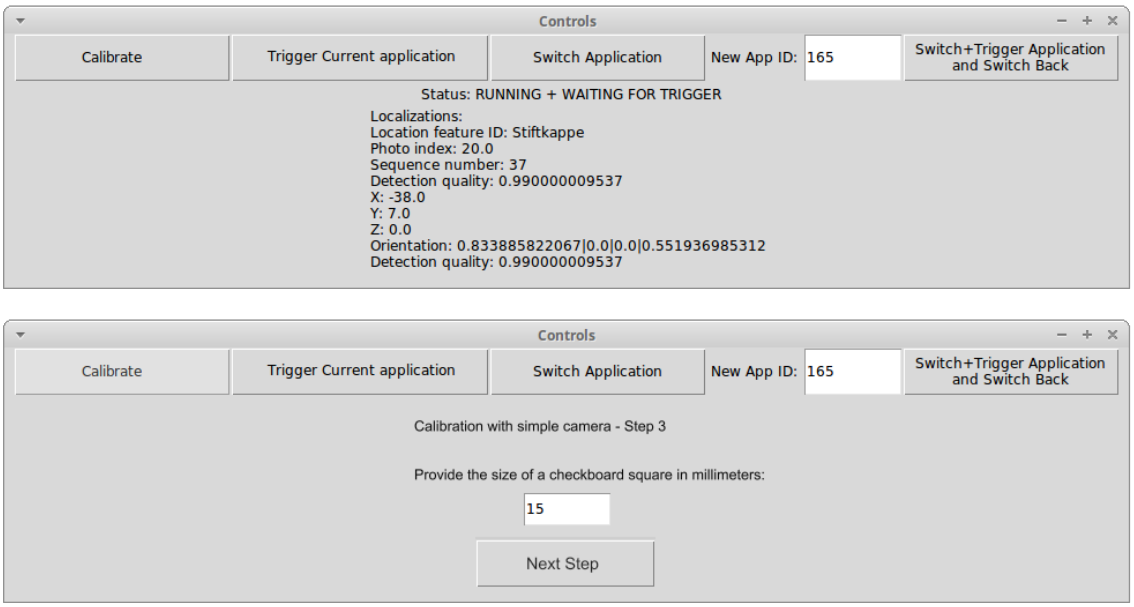

Figure 3.14: The tKinter front-end. After triggering a localization (above) and while waiting for an input value during calibration (below).

Chapter 4

Evaluation

Part of the process to achieve the goals set for a software project is to evaluate the results. This has been performed for the two major blocks, the SmartCamera Driver and the Camera Calibration and that is described within this chapter.

4.1 SmartCamera Driver

At the time of thesis submission, the SmartCamera Driver following the interface designed in section 2.1 SmartCamera Driver and implemented as in section 3.1 BVS002C Driver has been used in two installations at the drag & bot GmbH. Those have been practical tests of the implementation. Additionally, the transition from one installation to the other allowed an insight into the quality of the interface with respect to the company's Goals and Research Question.

4.2 SmartCamera Driver at Industry Fair (SPS IPC Drives Nürnberg)

First practical usage was a project initiated by the Balluff GmbH that provided the SmartCamera BVS002C. It has been presented to the public on the industry fair "SPS IPC Drives"[1]. The installation consisted of a public accessible table with the SmartCamera mounted above, so its detection-area contains a marked region on the table with a size of approximately 30cm width and 25 cm height. A set of tangram pieces made of wood was provided for the visitors of the fair to place tangram figures with. Additionally there was a robot cell with an "Universal Robot 3" equipped with 3D printed suction cup gripper. The cell also contained a table with fixtures for all parts of a secondary tangram set.

[1] cf. Nue, whole.

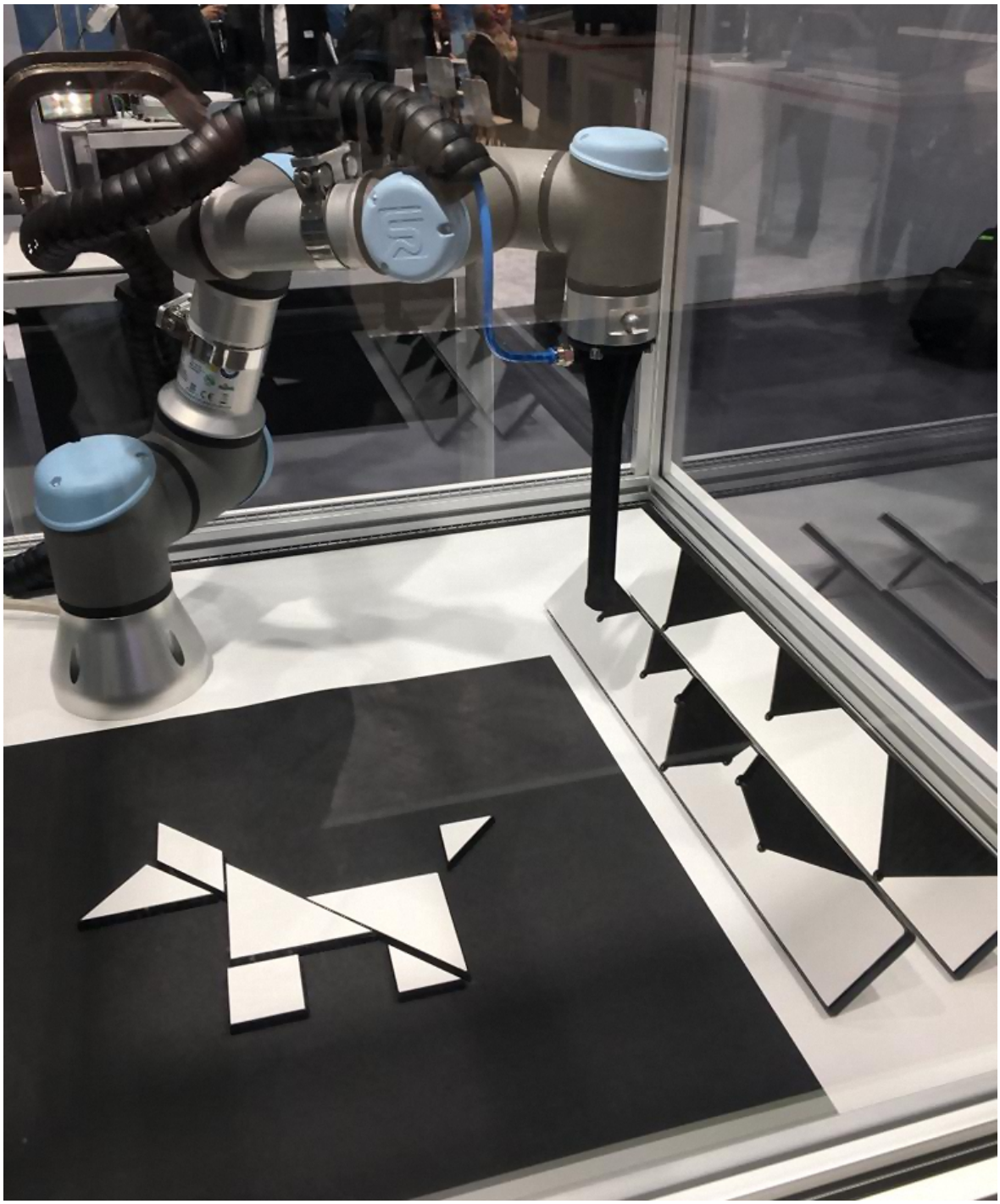

Figure 4.1: Installation for the SPS IPC Drives fair

The task to fulfill by this setup was a demonstration of the localization algorithm of the SmartCamera. Whenever a visitor of the fair placed a new tangram figure, the machine vision system should localize every part of the tangram, transmit their relative positions and the robot should pick up the corresponding parts from the fixtures to assemble the same figure the visitor had made. Effectively the robot copies the human actions. This can be seen in action in the video 4.2.

https://www.dropbox.com/s/00forx7yrtylcfn/DnB%20Tangram.avi?dl=0

Furthermore, the screenshot of the preview-window 4.2 shows the shapes of objects placed by the user. However those shapes have been hardcoded into the implementation because the SmartCamera can only transmit the feature-id and position of localized objects, not their actual shape or contours.

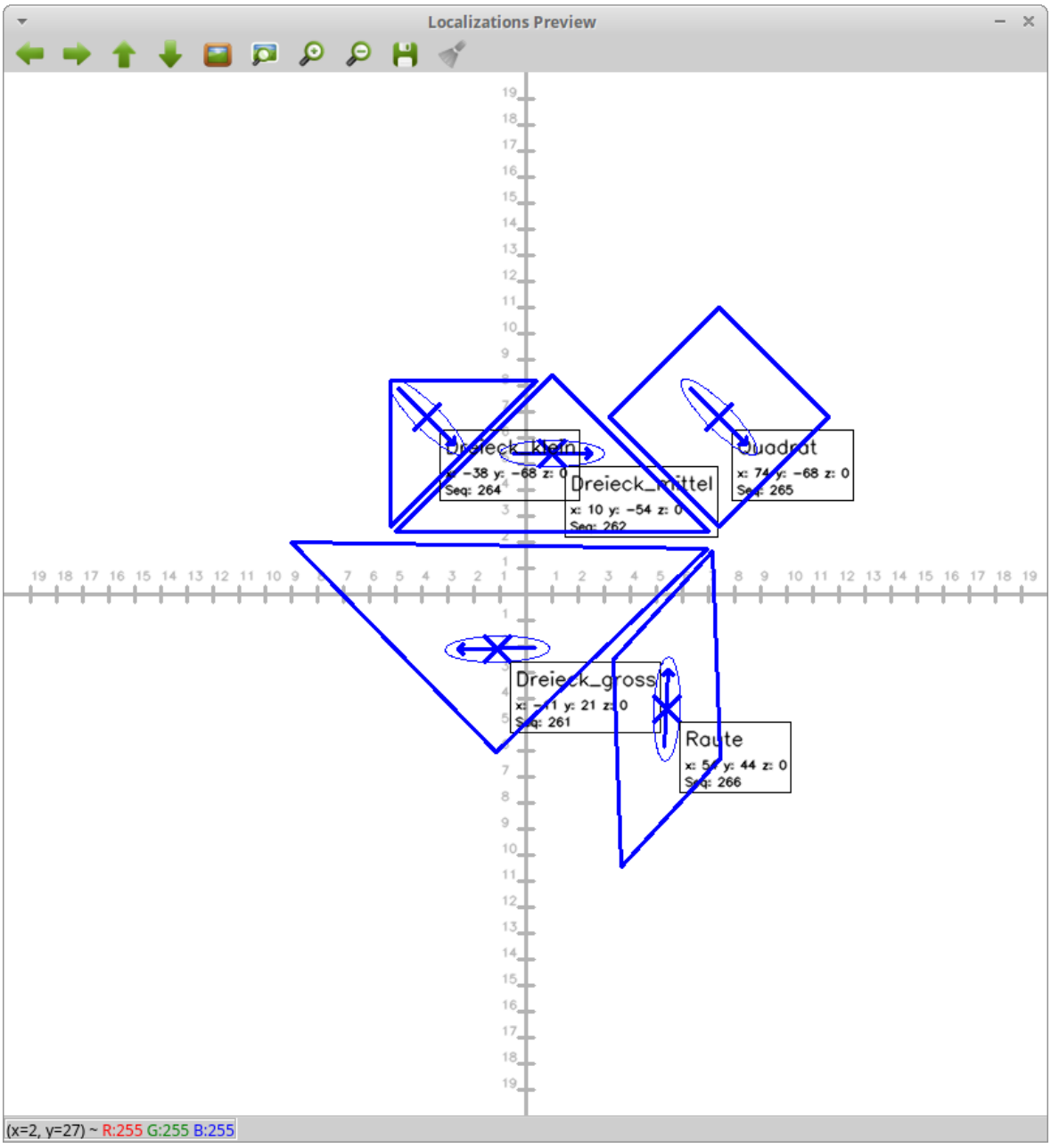

Figure 4.2: Preview-Window Showing the Sshapes of a Tangram.

The SmartCamera Driver proved to be usable for this task as it provided the localizations of all parts detected by the camera and that output could be processed in two specifically written function blocks to iterate through all parts and for each compute a series of movement commands for the robot.

4.3 Pickup & Sorting Demonstration

The drag & bot GmbH possess a miniature industrial robot, "Meca500"[2] that is equipped with a two centimeter large gripper and built together with an industrial computer into a mobile suitcase (seen in 4.3). It is used as demonstration system for the capabilities of the drag & bot software. Since machine vision is desired to become part of the company's offers to customers (see section 5.1 The Future), a presentable installation onto the Meca suitcase was needed that provides a fixture for the Balluff SmartCamera BVS002C. This was an occasion to evaluate the usage of the SmartCamera driver together with the camera calibration system.

Figure 4.3: The Meca500 on its Suitcase with the Balluff SmartCamera above

The demonstration that has been designed and programmed with function blocks uses the camera to localize two different types of small objects, then pick them up one after another and drop them into two different containers, depending on their type.
After this demonstration has been showcased and reprogrammed by participants of

[2] cf. Mec, whole.

an in-house robotics hands-on training, it has been concluded that the machine vision interfaces fulfill their purpose and the usage is adequate for the target customers and users of drag & bot.

4.4 Testing: Edge Cases of Calibration

The projects mentioned above and all physical setups of the Camera Calibration that had been installed during the time of writing this thesis, involved a calibration-plane that was either completely horizontal or only slightly skewed. During first testing during the implementation it turned out that overall wrong formulas could result in correct results when robot and calibration-plane had certain relative orientations to each other. This was usually the case when a coordinate flipped from the positive value range (0 to ∞) to the negative value range (0 to -∞). The conclusion was that a specially structured testing is required to ensure that all possible constellations are supported correctly by the camera calibration system.

In three-dimensional space, eight quadrants have been selected to test calibration in. Each with one of the three coordinates differing in their sign (positive or negative). The tests were performed with drag & bot using a simulator provided by the company Universal Robots instead of the real robot. This allows to test applications without requiring the physical robot and that system has been used for evaluation processes as it speeds up development and enables multiple software development projects to work simultaneously without multiple robots. Effectively this allowed to perform the calibration as if real, but using a rendered 3D-view that shows the robot like seen in 4.4.

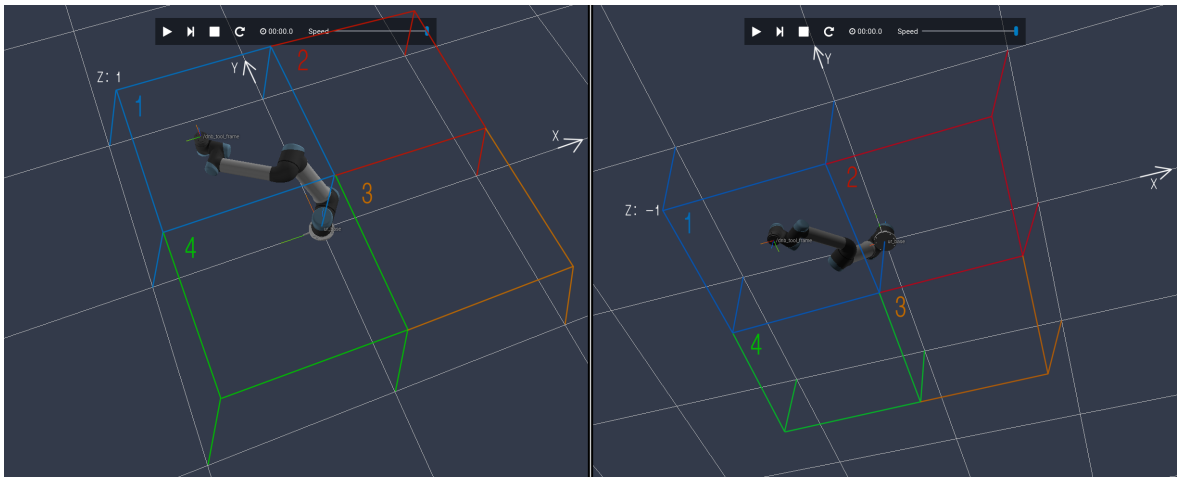

Figure 4.4: The Universal Robot simulated and rendered. Left: In quadrant 1 (z ¿ 0) Right: In quadrant 5 (z ¡ 0)

The calibration has been performed twice for every quadrant and checked for correctness by comparing computed values with expected values as well as the expected vi-

sual result. Aside from verification of the mathematical calculations (whose in-depth proving would have exceeded the scale of this thesis), this practical, yet simulated testing served as an evaluation of the soundness of the implementation.

Chapter 5

Conclusions and Outlook

The design and implementation work that lead to this thesis indicates that an interface for machine vision field that fulfills the goals of reusability and extensibility is possible. Using this interface it has been achieved that a new feature for the SmartCamera has been implemented within half a work day. Furthermore, for example structural changes of the calibration process resumes to changes in a single file, not requiring to alter code that belongs to the front-end.
Therefore the paper has shown how this kind of project can be developed at least for the software drag & bot, without adding external dependencies within the interface aside of the surrounding framework. The conditions are that the subfields of machine vision the interface portrays (in the particular example, SmartCameras and Camera-Calibration), are sufficiently analyzed to find mutuality between different manufacturers.
A limitation lies in achieving a maximum of universality while using a fixed framework. With long-term-use and re-use in mind designed algorithms like the "sequence system" can save implementation time in particular sections of the software, like in this case, in the front-end.

5.1 The Future

Since machine vision is a growing market (see subsection 1.1.1 Purpose and Market) this becomes an increasingly important field for the customers of the drag & bot GmbH and therefore for the company itself as well. As of now, five fields of activity are planned or started:

Implementation of the Calibration into Front-End Possibly using the "sequence system", the calibration will be implemented into the front-end of drag and bot to replace the provisional tKinter based user-interface that has been implemented during the development process of this thesis (see 3.2.8).

Support of a SmartCamera by Cognex As of the time of this thesis, a SmartCamera from the In-Sight 7000 series shall be ordered. A driver using the Extensible SmartCamera Interface shall be developed soon and thus the camera will be usable with drag & bot.

Development of Function Block Based machine vision Library For comparatively simple machine vision tasks, where sufficiently qualitative algorithms are publicly available like pattern-matching, (bar-)code reader, presenceverification etc., a set of function blocks shall be developed that utilize direct image input from non-smart cameras. This shall enable the company to offer machine vision solutions without requiring the customer to buy or own a SmartCamera.

Cooperation with Basler Multiple series of the 2D cameras developed by Basler shall be usable with drag & bot. Those should become configurable through another universal interface and the camera output shall be usable with the function blocks based library.

Eventually, the pickup & sorting demonstration on the Meca500 could be advanced by involving another project that has been developed recently within the company: A palletizing system. Using this, the program could place the sorted objects onto accurate positions instead of just dropping them into a bin.

Chapter 6

Attachments

Because not all sources are publicly accessible, two files have been attached to this thesis within the separate directory "Attachments". The files are:

Cognex IN-SIGHT 2000 Overview A document from the COGNEX homepage only accessible with an account (source within the bibliography).

LumiTrax Vision Systems Proposal A document from the Keyence homepage only accessible with an account (source within the bibliography).

Glossary

3D See "three-dimensional".. 10, 11

application
In the context of the Balluff BVS002C SmartCamera this refers to a user-designed program that has been developed through the browser-interface of the camera. It determines what machine vision tasks are performed. Complete applications can either be triggered through software externally or are running in an endless loop automatically.. 41, 42

back-end
The server structure of a software. Here: Usually the server of drag & bot that handles account data, licences and function blocks as well as programs made of function blocks.. 21

calibration-plane
A pane that has been computed by calibration. Its normal matches the axis of the camera's objective.. 64

computer vision
Encompasses technology, hard- and software that enables digital systems to perceive the environment optically.. 7

CPU
Central Processing Unit: Usually refers to the processor in desktop computers but also in some industrial devices. For common CPU architectures like ARM and X86, compilers and interpreters for many programming languages exist. This results in high versatility.. 12

end-user
The person or people who use a software.. 21, 23

ethernet
Basic protocol used in LAN.. 15

execution-time-predictability
How accurately the duration of a certain procedure can be specified.. 25

FPGA
Field Programmable Gate Array: An integrated circuit that can be programmed on gate-level and then perform similarly to a chip that has been directly made like that in silicon. Highly specialized programming languages and hard-ware-close design make its usage complex.. 13

frame
See "ROS-frame".. 15, 32

front-end
The visual interface of a software. Here: The browser-interface of drag & bot.. 21–23, 30–32, 34, 37, 50, 59, 66

function block
Element of a program in drag & bot. Consists of input and output parameters as well as Python code that utilizes the parameters.. 21, 37, 38, 40, 43, 44, 62, 63, 67

GigE
Gigabyte Ethernet. A network (or direct connection) based on ethernet with a speed of at least 1Gbit/s.. 17

Graphic card
Usually a computer component that is responsible for rendering graphics. The integrated computing units can be re-purposed for massive parallel programming.. 12

hex format
The way of expressing numbers on base 16.. 42

IC Integrated Circuit. 13

industrial robot
A robot arm that is designed to withstand the requirements of an industrial environment.. 17

interface element
Encapsulated part of an interface. Here: Usually a ROS-service.. 25, 26, 32

little-endian
Encoding method: Lower bytes first.. 42

local network
LAN: An ethernet-based network between computers connected only with switches in-between.. 15

machine vision
Subsection of computer vision. Revers to application in industrial environments usually to provide information to control production installations like robots.. 3, 4, 7–9, 11–13, 15–20, 22, 24, 37, 38, 40, 41, 61, 63, 64, 66, 67

MQTT protocol
A lightweight writer/subscribe protocol common in industrial systems as well as IOT devices.. 15

node
In this context, refers to ROS-node. 14

observation area
The area that is within the viewport of a camera that belongs to a machine vision system.. 10, 11, 16

operation tool
The device or tool attached to the end of a robot-arm. Usually allows interaction. For example grippers. An inspection-camera is another possibility.. 16

publishing
Sending/Writing data to a ROS-topic.. 15

robot-cell
A sturdy cage encasing an industrial robot. Usually it prevents human access for safety reasons. Additionally it allows to attach equipment to its frame.. 16, 56

robot-system
Here: The control-software of drag & bot that consists of components capable of accessing the hardware of the installation (for example the robot or SmartCameras).. 32

ROS
Robot Operating System. A framework for robots.. 14, 26, 32

ROS Master
The core of the ROS framework. It always starts first and controls the ROS-nodes and ROS-topics.. 15

ROS-frame
A coordinate system within ROS. For example the base of a robot has a frame and so has the TCP.. 56

ROS-message
A defined datastructure that can be used in ROS-topics and ROS-services.. 43–45

ROS-node
A process that has a name for identification and is part of a ROS-runtime.. 41, 47

ROS-service
An interface element of the ROS framework. Consists of parameters and response values and can be called similarly to a class method but through the ROS framework.. 50

ROS-topic
ROS-internal datastructure ROS-nodes can write (publish) on. Subscribing nodes can read.. 43

Shape-from-Shading
A technique to achieve three-dimensional information about an object with a single camera but multiple light sources. The information can be deduced from how light is reflected when it comes from different directions and thus causes different shades.. 11

Smart Camera
See "SmartCamera".. 11

stateful
An interface concept that means that the device holds an internal state and therefore the elements of the interface cannot be used independently from each other.. 24

string
Usually a datastructure consisting of a list or array of characters.. 14

subscribed
A hook has been set to execute when new data has been written to a ROS-topic.. 15

three-dimensional
Data with three dimensions. Usually X, Y and Z. For example, the position of a robot's TCP in the world.. 30, 38, 51, 55, 56

ToF
Time of Flight refers to a technique to generate distance-information by measuring the time light has taken from the source to the sensor.. 10

topic

Refers to ROS-topic.. 15

transform

Transferring coordinates or orientations from one coordinate-system to another. 17

two-dimensional

Data with two dimensions. Usually X and Y. For example, a location on an image.. 30, 46, 55, 56

workpiece

Objects that are produced or altered as part of a manufacturing process.. 8

Bibliography

[All] Adam Allevato. *ROS Wiki: Names*. (Accessed on 02/11/2019).

[Bas] Basler. *Time of Flight camera*. (Accessed on 02/12/2019).

[COG] COGNEX. *Introduction to Machine Vision*. https://www.assemblymag.com/ext/resources/White_Papers/Sep16/Introduction-to-Machine-Vision.pdf. (Accessed on 10/30/2018).

[Coga] Cognex. *3D-A5000 SERIES AREA SCAN 3D CAMERA*. (Accessed on 02/12/2019).

[Cogb] Cognex. *Visionpro VIDI Deep learning - based software for industrial image analysis*. (Accessed on 02/17/2019).

[com] Camera Link committee. *Camera Link*. (Accessed on 02/11/2019).

[db] drag and bot. *drag and bot: Use industrial robots like a smartphone*. (Accessed on 02/17/2019).

[Deb] Jan Debiec. *ROS Wiki: Multiple Machines*. (Accessed on 02/11/2019).

[DG17] G D'Emilia and D Di Gasbarro. „Review of techniques for 2D camera calibration suitable for industrial vision systems". In: *Journal of Physics: Conference Series* 841 (May 2017), p. 012030. DOI: 10.1088/1742-6596/841/1/012030. URL: https://doi.org/10.1088%2F1742-6596%2F841%2F1%2F012030.

[FLI] FLIR. *Machine Vision Interface Comparison and Evolution*. (Accessed on 02/11/2019).

[Fou] Open Source Robotics Foundation. *About ROS*. (Accessed on 02/11/2019).

[How] Joseph Howse. *OpenCV Computer Vision with Python*. (Accessed on 02/26/2019).

[IMP] AI IMPACTS. *Recent trend in the cost of computing*. (Accessed on 02/26/2019).

[Jue15] Christian Frese Juergen Beyerer Fernando Puente Leon. *Machine Vision: Automated Visual Inspection: Theory, Practice and Applications*. Summer 2015.

[KEY] KEYENCE. *CV-X Series Intuitive Vision System Ver.4.2 Digest version of catalogue*. (Accessed on 02/13/2019).

[Key] Keyence. *LumiTrax*. (Accessed on 02/13/2019).

[Koc09] Kevin Kocher. *Finding Possible Solutions to Avoid Future Deficient Covers of Swiss Pension Funds*. Aug. 2009. URL: http://www.fhnw.ch/wirtschaft/dienstleistung/studierendenprojekte/olten/bisherige-projekte/bachelor-thesis-2009/finding-possible-solutions/finding-possible-solutions.

[KUK] KUKA. *KUKA Handheld*. (Accessed on 02/17/2019).

[Mau] Dr. Karl-Heinz Maurer. *Sperrvermerk (Vertraulichkeitserklärung) für die Masterarbeit*. (Accessed on 02/13/2019).

[Mec] Mecademic. *Meca500 - Ultracompact six-axis robot arm*. (Accessed on 02/21/2019).

[Nue] Nuernbergmesse. *SPS IPC Drives*. (Accessed on 02/21/2019).

[Ope] OpenCV. *OpenCV Implemenetation of Checkerboard-Detection (among others)*. (Accessed on 02/26/2019).

[pen] pentadecagon. *How to convert direction vector to euler angles*. (Accessed on 02/26/2019).

[Rep17] Cooked Research Reports. *Machine Vision Market Research Report Forecast to 2022 — MRFR*. (Accessed on 10/30/2018). May 2017.

[Res] Inkwood Research. *GLOBAL MACHINE VISION MARKET FORECAST 2018-2026*. (Accessed on 02/07/2019).

[Rie02] Wolf-Fritz Riekert. *Eine Dokumentvorlage für Diplomarbeiten und andere wissenschaftliche Arbeiten*. Apr. 2002. URL: `http://v.hdm-stuttgart.de/~riekert/theses/thesis-arial11.doc`.

[Rob] Universal Robots. *UR Controller*. (Accessed on 02/17/2019).

[Stu18] Landesmesse Stuttgart. *Vision: Trade Fair Impressions 2018*. (Accessed on 2/07/2019). Nov. 2018.

[Tea] The HiveMQ Team. *MQTT Essentials Part 2: Publish & Subscribe*. (Accessed on 02/27/2019).

[Tre] Brandon Treece. *CPU or FPGA for image processing*. (Accessed on 02/12/2019).

[Vis18] Maximilian Visotschnig. *Studienarbeit: Bildverarbeitung in drag and bot*. Internally for the drag and bot GmbH. May 2018.

[Wen15] Anne Wendel. *VDMA: Industrielle Bildverarbeitung bricht alle Rekorde - VDMA*. (Accessed on 10/30/2018). July 2015.