Thesis for the degree of Doctor of Philosophy

Flexible Composition of Robot Logic with Computer Vision Services

Oleksandr Semeniuta

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2018

Flexible Composition of Robot Logic with Computer Vision Services
OLEKSANDR SEMENIUTA
ISBN 978-91-7597-777-5

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 4458
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE–412 96 Göteborg, Sweden
Telephone + 46 (0) 31 – 772 1000

Typeset by the author using LaTeX.

Printed by Chalmers Reproservice
Göteborg, Sweden 2018

to my mother, Nadiia Semeniuta

List of Publications

This thesis is based on the following appended papers:

Paper 1 Oleksandr Semeniuta, Sebastian Dransfeld, Kristian Martinsen, and Petter Falkman. Towards increased intelligence and automatic improvement in industrial vision systems. *Procedia CIRP*, 67:256–261, 2018. ISSN 22128271. doi: 10.1016/j.procir.2017.12.209

Paper 2 Oleksandr Semeniuta, Sebastian Dransfeld, and Petter Falkman. Vision-based robotic system for picking and inspection of small automotive components. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 549–554. IEEE, aug 2016. ISBN 978-1-5090-2409-4. doi: 10.1109/COASE.2016.7743452

Paper 3 Oleksandr Semeniuta and Petter Falkman. EPypes: a framework for building event-driven data processing pipelines. Submitted to: *PeerJ Computer Science*

Paper 4 Oleksandr Semeniuta and Petter Falkman. Flexible image acquisition service for distributed robotic systems. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 106–112. IEEE, jan 2018. ISBN 978-1-5386-4652-6. doi: 10.1109/IRC.2018.00024

Paper 5 Oleksandr Semeniuta and Petter Falkman. Event-driven industrial robot control architecture for the Adept V+ platform. Submitted to: *Frontiers in Robotics and AI*

Other relevant publications co-authored by Oleksandr Semeniuta:

Oleksandr Semeniuta. Analysis of camera calibration with respect to measurement accuracy. *Procedia CIRP*, 41:765–770, 2016. ISSN 2212-8271. doi: http://dx.doi.org/10.1016/j.procir.2015.12.108

Oleksandr Semeniuta and Petter Falkman. Discrete event dataflow as a formal approach to specification of industrial vision systems. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, volume 2015-Octob, pages 849–854. IEEE, aug 2015. ISBN 978-1-4673-8183-3. doi: 10.1109/CoASE.2015.7294187

Ivanna Baturynska, Oleksandr Semeniuta, and Kristian Martinsen. Optimization of process parameters for powder bed fusion additive manufacturing by combination of machine learning and finite element method: A conceptual framework. *Procedia CIRP*, 67:227–232, 2018. ISSN 22128271. doi: 10.1016/j.procir.2017.12.204

Acknowledgments

Big thanks to my supervisor, Petter Falkman, for lots of inspiration throughout these four years. I wish every PhD candidate had a supervisor as great as you.

I would like to thank the Norwegian Research Council and the MultiMat project for funding my work, and Håkon Raabe and Anja Solheim for their friendly support. Thanks to Geir Liaklev and Henning Rud from Kongsberg Automotive AS for sharing their vision and industrial knowledge, and for giving me freedom in shaping my research.

Big thanks to the Production Technology department at SINTEF Raufoss Manufacturing AS, and specifically the Raufoss crew: Sebastian Dransfeld, Ådne Solhaug Linnerud, and Mats Larsen. I have learned a lot from you.

Thanks to my colleagues at the NTNU's Department of Manufacturing and Civil Engineering. Thanks to Kristian Martinsen for the support throughout all my PhD years, and especially for the laboratory equipment arrangements. Ability to work with the Adept robot in the last phase of my PhD research had greatly streamlined my activities. Thanks to Torbjørn Skogsrød and Iver Eugen Jensen for great leadership assistance in tough situations. Additional gratitude to Torbjørn for believing in the importance of my research and my ideas. Thanks to Michael Cheffena for the interesting conversations about the future of industrial communication. Thanks Chunhong Luo and Natasja Bours for their top-notch administrative assistance.

Thanks to all members of the Automation research group at Chalmers, especially Martin Fabian, Bengt Lennartson, Knut Åkesson, Kristofer Bengtsson, Ashfaq Hussain Farooqui, Martin Dahl, and Sarmad Riazi.

Thanks to my friends, Anastasiia Moldavska, Christoffer Vargtass Hallstensen, Alla Marchenko, Ruslan Zhuravchak, Andrii Shalaginov, Marina Shalaginova, Jo Sterten, Eivind Arnstein Johansen, and Therese Kristin Havnsund.

Tremendous thanks to my beloved Ivanna for the greatest support one can get.

Oleksandr Semeniuta, Göteborg/Gjøvik, August 2018

Acknowledgments

Big thanks to my supervisor, Petter Falkman, for lots of inspiration throughout these four years. I wish every PhD candidate had a supervisor as great as you.

I would like to thank the Norwegian Research Council and the MultiMat project for funding my work, and Håkon Raabe and Anja Solheim for their friendly support. Thanks to Geir Liaklev and Henning Rud from Kongsberg Automotive AS for sharing their vision and industrial knowledge, and for giving me freedom in shaping my research.

Big thanks to the Production Technology department at SINTEF Raufoss Manufacturing AS, and specifically the Raufoss crew: Sebastian Dransfeld, Ådne Solhaug Linnerud, and Mats Larsen. I have learned a lot from you.

Thanks to my colleagues at the NTNU's Department of Manufacturing and Civil Engineering. Thanks to Kristian Martinsen for the support throughout all my PhD years, and especially for the laboratory equipment arrangements. Ability to work with the Adept robot in the last phase of my PhD research had greatly streamlined my activities. Thanks to Torbjørn Skogsrød and Iver Eugen Jensen for great leadership assistance in tough situations. Additional gratitude to Torbjørn for believing in the importance of my research and my ideas. Thanks to Michael Cheffena for the interesting conversations about the future of industrial communication. Thanks Chunhong Luo and Natasja Bours for their top-notch administrative assistance.

Thanks to all members of the Automation research group at Chalmers, especially Martin Fabian, Bengt Lennartson, Knut Åkesson, Kristofer Bengtsson, Ashfaq Hussain Farooqui, and Martin Dahl.

Thanks to my friends, Anastasiia Moldavska, Christoffer Vargtass Hallstensen, Alla Marchenko, Ruslan Zhuravchak, Andrii Shalaginov, Marina Shalaginova, Jo Sterten, Eivind Arnstein Johansen, and Therese Kristin Havnsund.

Tremendous thanks to my beloved Ivanna for the greatest support one can get.

Oleksandr Semeniuta, Göteborg/Gjøvik, August 2018

Abbreviations

AGV	Automated guided vehicle
API	Application programming interface
AMQP	Advanced Message Queuing Protocol
ARM	A family of reduced instruction set computing architectures for computer processors
AS	Aksjeselskap (Norwegian for *limited liability stock-based company*)
CAD	Computer-aided design
CERN	The European Organization for Nuclear Research
CPS	Cyber-physical system
CPU	Central processing unit
CSP	Communication Sequential Processes
DAG	Directed acyclic graph
DDS	Data Distribution Service
DEDF	Discrete event data flow
FAS	Flexible assembly system
FIFO	First in, first out
FN	False negatives
FP	False positives
FPGA	Field-programmable gate array
FxIS	Flexible image acquisition service
GCC	GNU Compiler Collection
GPU	Graphical processing unit
GUI	Graphical user interface
HTTP	Hypertext Transfer Protocol
I/O	Input/output
IOCP	Input/output completion port
KA	Kongsberg Automotive AS
LAN	Local area network

LED	Light-emitting diode
LESH	Local energy-based shape histogram
LISA	Line Information System Architecture
ML	Machine learning
MLE	Maximum likelihood estimation
MSE	Mean squared error
MQTT	Message Queuing Telemetry Transport
NTNU	Norwegian University of Science and Technology
PC	Personal computer
PLC	Programmable logic controller
POSIX	Portable Operating System Interface
RGB	The red/green/blue color model
RMS	Root mean square
ROS	Robot Operating System
RPC	Remote procedure call
RS-232	A standard for serial transmission of data
$SE(3)$	Special Euclidean group
SIFT	Scale-invariant feature transform
SDF	Synchronous data flow
$SO(3)$	Special orthogonal group
SOA	Service oriented architecture
SOAP	Simple Object Access Protocol
SRM	SINTEF Raufoss Manufacturing AS
SURF	Speeded up robust features
SVD	Singular-value decomposition
SVM	Support vector machine
TCP	Transmission control protocol
TN	True negatives
TP	True positives
UDP	User datagram protocol
UML	Unified Modeling Language
USB	Universal serial bus
XML	Extensible Markup Language
VLAN	Virtual local area network
YARP	Yet Another Robot Platform
ZMTP	ZeroMQ Message Transport Protocol

Contents

Part I

Introductory chapters

Chapter 1

Introduction

The increased demands for flexibility and quality stimulate manufacturing companies to seek production solutions that are both easy to adapt to changing needs and able to consistently produce parts in accordance with the specification. Automation is in many cases regarded as an indisputable means of realization of such production systems, particularly in high cost countries.

Automated solutions for high-volume production often constitute dedicated high-speed machines and transfer lines, especially when dealing with small parts and products. Practically, such systems, which can be categorized as *hard automation*, possess high speed at the expense of limited flexibility. In contrast, or rather in addition, to them, industrial robots play the role of a canonical workhorse of manufacturing automation. Not only robots are general-purpose and flexible, but with effective integration of sensory data, one can tackle problems with increased uncertainty and harness the knowledge about the surrounding environment.

In today's industrial automation, systems based on programmable logic controllers (PLCs) have become pervasive. PLCs allow for implementing control systems using familiar graphical programming abstractions, extensive I/O interfaces, and high-frequency real-time control capabilities. PLC vendors have been constantly adding higher-level functionality, such as connection to product lifecycle management systems, virtual manufacturing environments, and other enterprise information systems. PLCs are particularly suitable for implementation of sequential control systems, which are ultimately based on discrete event logic.

In addition to industrially accepted PLCs, two other competing categories of control devices are gaining increased popularity, those being embedded systems and general-purpose computers. The former, typically based on microcontroller technology and ARM architecture, has long been a part of many consumer products, and, with the proliferation of the novel smart manufacturing paradigms such as Industry 4.0, are becoming more common in the industrial automation environments. For complex signal processing, FPGAs are often used as a means of building embedded systems with high performance computations. On the other side of the spectrum, general-purpose computing has been a backbone for complex processing tasks, which, in relation to automated manufacturing, encompass signal processing, computer vision, machine learning etc. The current trend is to harness graphical processing units, computing clusters, and cloud computing environments.

Inspired by biological agents, which possess rather sophisticated perception function, many researchers have been working for years on developing methods for the realization of perception in man-made systems. This has led to methods of sensor data acquisition and processing in order to get insights into the surrounding environment. In manufacturing systems and the associated industrial robotics, perception based on *artificial vision* plays a key role as it leads to a number of important capabilities such as automatic estimation of product quality, volumetric properties of the surrounding environment, specifics of physical dynamics of system operation etc.

Similarly to the visual function in biological agents, which is heavily tied with cognitive processing, artificial vision systems are naturally considered in conjunction with methods from artificial intelligence and machine learning. These methods are aimed at mimicking the biological cognitive function, and are based on the combination of probabilistic and statistical modeling, optimization methods, advanced algorithms and data structures, and implementation of clever high-performance computing architectures.

As such, computer and machine vision are ever-growing fields with solid acceptance in the industry and a large number of useful applications. Modern computer vision includes sophisticated methods for robust feature detection such as SIFT, SURF, HoG, offering scale and orientation invariance, and thus allowing for reliable matching of interest points in several views and, as a result, reconstruction of the scene in 3D. This has led to many useful applications in robotics, based on point clouds obtained from matched features. Hardware solutions for 3D image acquisition has also become widely-used, from consumer-grade devices based on structured light, to industrial high-density laser scanners. The increased availability of computational resources in recent years, combined with greater availability of data, streamlined the application of machine learning methods, particularly deep learning, for analysis of visual data. Nevertheless, a large number of challenges exists in developing vision algorithms, including a high level of uncertainty in image data and the environment, properties of illumination and reflection, processing time limitations, and the like.

Naturally, the first intention when developing any vision algorithm is *feasibility*:

whether the pilot implementation can prove that a particular vision task can be algorithmically solved. Further, it is important to assure higher accuracy, robustness to various sorts of deviations, and reasonable processing time. At the same time, when an industrial solution is being developed, it is essential to perform system integration and hardware-in-the-loop testing even in the early stages of system development.

In the context of industrial automation, the vision function has a direct relation to the physical behavior of the larger system. As an example, even a relatively simple robotic cell constitutes a distributed collection of mechatronic entities that deliver their value collectively. As shown in Figure 1.1, a typical machine vision system is functionally comprised of such processes as imaging (with the optical system and an image sensor), acquisition (transforming captured images to a computer memory), and image processing with high-level decision making (both being implemented with software components, but operating at different levels of abstraction). In a way, the traditional approach of vision systems development focuses primarily on the image processing and decision-making tasks, mainly due to the abovementioned feasibility requirement. However, for effective integration with the process/scene and other system components, which are typically distributed, a wider approach should be taken, with an explicit goal of bridging the functions of *perception* and *action*.

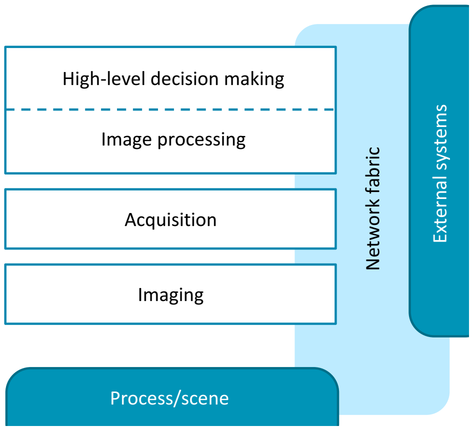


Figure 1.1: A layered view of a machine vision pipeline

Biologic agents naturally possess an effective interplay between perception and action, as well as their relation to planning and other types of complex cognitive processing. When it comes to automation systems, the tasks of geometric calibration, unification of coordinate systems, pose estimation, and visual servoing specifically target coupling the vision function with the rest of the system. Each of these tasks are associated with certain isolated aspects of implementation of visually-guided behaviors. What is missing, however, is an established system-level methodology of vision function integration into automated systems.

The work underlying this thesis combines the problematics of machine vision algorithms development with their integration into larger systems possessing event-driven behavior. The thesis focuses on composition of three functions: (1) robot logic, (2) image acquisition function, and (3) image processing function. Each of the three is considered in details to achieve event more granular composition on the function level.

1.1 Research questions

This thesis is aimed at answering the following research question.

RQ1 How to better couple the automated systems' perception and action with the means of computational abstractions and architectural solutions?

This research question is aimed at combining research directions that have traditionally been orthogonal to each other, though highly complementary in practice. The task of perception is typically concerned with signal analysis, and the implementation of the perception function is normally done with little consideration of how the dynamics of the entire system is going to be controlled. In this thesis it is assumed that one can do better with coupling perception and action in automated systems with the means of suitable modeling abstractions and software architectures.

RQ2 How can the image acquisition function be exposed in a service-oriented manner?

Image acquisition is present in every vision system, although it is normally not exposed to a computer vision developer in a flexible service-oriented way. This limits possibilities for creation of modular distributed systems that utilize vision sensing. This thesis aims at creating a practical realization of its ideas, including a flexible image acquisition service with well-understood dynamics.

RQ3 How to achieve greater composability of communication-heavy robot logic by utilizing discrete event nature of distributed robotic systems?

Programming industrial robots directly in the robot controller is a standard practice that leads to efficient system operation. At the same time, the resulting robot logic is often rigid and difficult to integrate into a distributed system with varying communication modalities. It is therefore of interest to investigate the capabilities of a high-level robot control node that would allow seamless composability of communication-heavy logic units.

1.2 Contributions of the thesis

This thesis describes the following contributions:

1. Study of the specifics of the production systems of the case company (Kongsberg Automotive AS) with focus on (1) industrial application of machine vision system to the current challenges the company is facing and (2) possible vision-based solutions that could be implemented in the future.

2. Implementation of an image analysis pipeline, including feature engineering and machine learning, for identification, classification and close-range measurements of star washer components in a robotic cell.

3. Development of a software framework for implementation of event-based data processing pipelines, aimed at application in automated systems with a strong perceptual component and publish/subscribe communication architecture.

4. Development of a flexible image acquisition service for use as a part distributed robotic systems with event-based communication; the service realizes continuous acquisition from multiple cameras and on-demand retrieval of images that are closely synchronized in time, both between the cameras and with the request event.

5. Formulation of an event-driven industrial robot control architecture for the Adept V+ platform, along with the associated software implementation, grounded upon (1) the robot controller providing a TCP/IP server and a collection of robot skills, and (2) a high-level control module deployed to a dedicated computing device, with an AsyncIO-based composable logic.

Chapter 2

Method

2.1 Methodological considerations

A common aspect of research in both robotics and computer vision is a strong focus on *feasibility* of solutions. Because in general it is challenging to implement a functional prototype, be it a robotic task, or a recognition algorithm, the particular implementation in itself possesses a high value. Thus, it makes sense not to do premature optimization in the early phases of system development, and focus on proving feasibility of automation of a task of interest. Exactly this focus is highlighted by Goldberg [7] as a principal difference between the areas of research in *robotics* and *automation*. Though very interconnected, there are principal differences in both the purpose of the research and the methodological approaches. While robotics, as noted before, emphasizes feasibility, automation emphasizes efficiency, productivity, quality, and reliability.

In relation to the dichotomy above, it is interesting to consider the classification of philosophical paradigms in the computer science community. As described by Eden [8], one can distinguish between three paradigms: rationalist, technocratic, and scientific. In short, the discussion comes down to understanding the nature of a computer program. The rationalist paradigm treats a program as a mathematical object one can formally reason about, the technocratic — as an engineered artifact one can test, and the scientific — as a natural phenomenon one can explore experimentally. It is noted that the scientific paradigm is typical in the fields of machine learning and robotics, mainly due to highly stochastic nature of the developed systems.

The emergent field of cyber-physical systems (CPS) has a strong focus on a rigorous approach to system design, aiming at combing formal models of physical dynamics with formal models of computational and communication systems. Thus, in general the CPS approach is highly rationalistic. At the same time, according to the CPS development framework formulated by Lee and Seshia [9], see Figure 2.1, rigorous modeling and analysis should be interleaved with the practical design process. The latter puts an emphasis on computing equipment mechanisms, software design, and input/output interface.

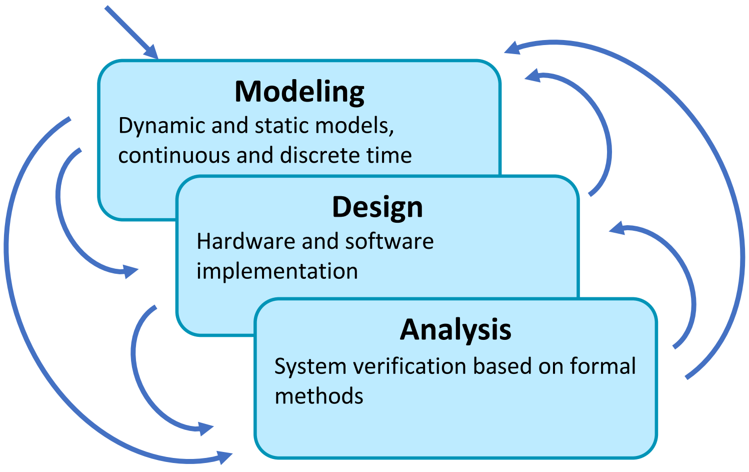

Modeling
Dynamic and static models, continuous and discrete time
Design
Hardware and software implementation
Analysis
System verification based on formal methods

Figure 2.1: Cyber-physical systems development framework (adapted from [9])

The advent of data science is characterized by the proliferation of ad-hoc approaches to scientific discovery, focusing on quick prototyping with the help of flexible programming tools. The same holds for feasibility-oriented research practices in computer vision and robotics. This may seem opposite to the structured engineering approaches and formal system analysis and synthesis. As argued by Lee [10], an implement-and-test style of system design is considered ”hacking, not based on sound engineering principles”. This argument is similar to the one made by Eden [8] concerning the domination of the technocratic paradigm in contemporary computer science, and the shortcomings associated with it.

It is important to mention ideas that bring more structure into the data science workflow without compromising flexibility. The central idea here is *”data science is software”* [11], stressing that although ad-hoc processes are essential for prototyping and data discovery, the resulting codebase should be treated as a software product. This implies requirements for maintaining a well-structured and well-documented project and applying software engineering practices alongside exploration and prototyping. A related idea, driven by the need for greater reproducibility, concerns custom scientific software modules and data science projects as immutable software artifacts. This can be achieved by harnessing containerization technology and the DataOps philosophy.

The work underlying this thesis targets the early stage of system development, and thus gravitates towards implementation-centric approaches. At the same time, the goal of the proposed solutions is to unify prototyping with well-defined specification of system structure and behavior. More details on this follows in section 2.4.

2.2 The MultiMat project

There always exists a pool of problems that companies in various industries are facing. In manufacturing, among others, there are challenges associated with new product development and the requirements for production systems. In addition, one always tackles the problems of improvement and evolution of existing production systems. The abovementioned challenges go hand in hand with opportunities that stem from successful systems implementation and novel non-trivial applications. And it is undoubtedly beneficial to carry out new technological developments in a trans-disciplinary environment. In such a way, ideas from different disciplines and intellectual schools are likely to converge.

The PhD research described in this thesis is done as part of the Norwegian innovation project *MultiMat*, focusing on development of technical solutions for manufacturing of novel multi-material products. The primary MultiMat project objective is set on developing new products and production processes based on combinations of dissimilar materials with integrated and automated injection molding, joining and assembly. The project utilizes trans-disciplinary R&D efforts within product development, materials technology, injection molding, assembly, joining, and flexible automation.

The main stakeholder in MultiMat is Kongsberg Automotive AS (KA), whose core competence in the project lies within the area of product development. The latter is naturally tightly connected to materials science and manufacturing processes, most notably injection molding and joining. However, in the Norwegian settings, practical implementations of industrial automation solutions is a critical factor ensuring the required quality and cost savings.

The increasing need for innovative automation solutions in Norwegian manufacturing industry places high expectations on vision- and sensor-based robotics and a wider field of flexible automation. SINTEF Raufoss Manufacturing AS (SRM), an R&D partner in the MultiMat project, have been conducting the main automation-related activities, including development of a robotized cell for integrated injection molding and assembly of newly-developed products.

In addition to the activities at SRM, this PhD research was carried out with a central focus on the combination of machine vision and robotics to tackle industrial problems. Particular cases of such problems were to be drawn from the MultiMat project, and specifically from the product development efforts of Kongsberg Automotive AS. When it comes to vision systems, KA is particularly interested in vision-based quality control of small components (star washers, O-rings, clamp rings) and the development of systems for data acquisition, storage and analysis to cope with visual variation and assure system improvement. As such, the case-related parts of this thesis discuss the tasks of vision-based analysis of small safety-critical parts that are included in KA products. A great deal of practical undertakings underlying this thesis were conducted at the SRM facilities with the support from the members of the Production Technology department at SRM.

2.3 Main theme of the PhD research

Previous experience and preliminary planning of this PhD research placed a key focus onto the *systemic aspects of machine vision and robotics.*

For one thing, even relatively simple assembly cell arrangements possess a complexity that has to be managed. In order for a collection of computational and mechatronic components to work as a system, issues of concurrency, communication, synchronization, and time need to be dealt with. To increase capabilities of automated systems, the complexity needs to be increased even more, for example by introducing additional sensor systems and computational equipment. A typical way of tackling complexity is by careful architectural design emphasizing modularity. However, architectural concerns often remain overlooked, especially when a quick implementation of robot/vision system prototype is desired.

The pervasive penetration of digital technology into every area of human activity has been specifically evident in the recent years. In particular, mobile technology in combination with cloud computing has in many ways transformed established workflows. In this new setup, the value is delivered by well-designed distributed systems, comprised of powerful cluster- and cloud-based back-ends and energy-efficient smart devices. The success of such technologies has motivated applied researchers and policymakers to put a special emphasis on more effective integration of numerous heterogeneous systems of various scales (ranging from embedded devices to large computing clouds). Thus, harnessing these novel systems is an integral part of such strategic initiatives as Industry 4.0 [12], Industrial Internet [13], and Smart Manufacturing Systems [14].

Flexible and scalable server-side applications, particularly those in cloud environments, are the result of proliferation of open source software, such as Linux-based operating systems, software containerization technologies, cluster-based file systems, and the like. A great influence in the cloud computing industry has been the Unix philosophy [15], which is a loose collection of principles centered around simplicity, modularity and composability. The same culture is now driving the data science and machine learning communities, as well as constituting a major force of the DevOps and DataOps philosophies.

When it comes to robotics and automation, there is already a big community around open source projects such as Robot Operating System (ROS), OpenCV, Point Cloud Library, YaRP, and many others. These technologies are widely used in the research environment, but slowly adopted in industry. Nevertheless, this thesis makes a bet on the open source approach and the Unix philosophy principles. In particular, the idea of *composability* is central in most of the described practical undertakings.

2.4 Solutions development as a research method

Because the research area of this thesis is highly applied, it is natural to perform research by developing practical solutions and functional prototypes.

There is a challenge with this approach in that it becomes difficult to distinguish engineering and research. Referring to the computer science philosophical paradigms, described earlier in section 2.1, the said application-oriented approach is closest to the technocratic paradigm. At the same time, the common theme in this thesis is the focus on general principles rather that concrete technologies. These principles include the following:

- Events as explicit building blocks of distributed systems;
- Executable direct acyclic graphs for design of computer vision algorithms;
- Architectural separation of low-level robot logic in the from of skills from high-level communication-heavy logic in the form of coroutines;
- Service-oriented architecture approach to exposing image acquisition function;
- Internal multi-threaded design of the image acquisition service with a set of concurrent data structures;
- Composition of image acquisition and image processing via thread-based concurrent objects and blocking queues;

- Composition of communication-heavy robot logic by cooperative multitasking.

Apart from implementation of laboratory prototypes, a collection of open source software projects is conceived as a part of this PhD project:

EPypes A framework for building computer vision and general data processing algorithms in the form of executable computational graphs, with additional capabilities of exposing the resulting graphs as reactive pipelines. Written in Python with support for ZeroMQ messaging middleware.

FxIS A framework for creation of flexible image acquisition services based on one or more cameras performing continuous image capture. Written in C++ with a Python extension, having initial support for Allied Vision GigE Vision cameras via the Vimba SDK.

pyadept A Python library for realization of high-level robot control nodes for the Adept V+ platform with logic composable from AsyncIO coroutines.

AdeptServer A collection of V+ programs for realization of Adept robot skills and a TCP server, used in conjunction with `pyadept`.

As most of the projects are Python-based, it is important to describe the stack of Python libraries this thesis' codebase depends on. As Figure 2.2 shows, Python of version 3.6 has been used to develop the projects. The core libraries include OpenCV (providing computer vision routines), NumPy (handling multi-dimensional arrays), Pandas (data analysis with data frames), PyZMQ (binding for ZeroMQ, which is used as the primary publish-subscribe communication means), and Protobuf (allowing to work with the Protocol Buffers serialization format, used to communicate data over wire). The next layers include scientific computing libraries (SciPy, Scikit-learn, Scikit-image, and Pillow), and the libraries used for data and graph visualization (Matplotlib, NetworkX, PyGraphviz, and nxpd).

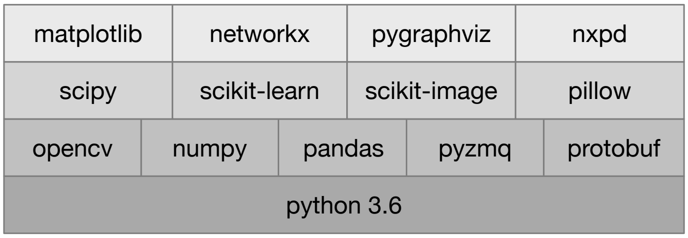


Figure 2.2: Python stack for the codebase of this thesis

The first C++ implementation of FxIS has been done on top of the Vimba SDK for Allied Vision cameras, with OpenCV providing abstraction for image handling. To produce a Python extension for FxIS that was further used together with EPypes, the Pybind11 library was used. CMake and GCC were used for building.

Laboratory work in connection with this PhD project was done with access to the following equipment:

- Adept Viper s850 industrial robot, Adept SmartController CX, Adept DeskTop programming software, communication over Ethernet with external systems.
- Anyfeed SX240 flexible feeder with RS-232 communication interface.
- Allied Vision GigE Vision cameras: Prosilica GC1020 (resolution 1024x768), Prosilica GC1350/GC1350C (resolution 1360x1024).
- Camera lenses: PENTAX C1614-M (focal length 8 mm), Fujinon HF35HA-1B (focal length 35 mm), TAMRON 25-HB/12 (focal length 12 mm).
- Raspberry Pi 3 Model B single board computers.
- Netgear GS108Ev3 managed Gigabit Ethernet switch.

As such, solutions development described in this thesis is done in a highly prototyping-oriented environment. Partly, this is attributed to a constrained selection of hardware available for experimentation, which didn't include such industrial-grade components as PLCs and fieldbus-based networks. As a result, the primary communication channel is Ethernet, with protocols based on TCP/IP. At the same time, similar setups are not uncommon in robotics research environments, motivated by flexibility of development and availability of general-purpose hardware for streamlining the prototyping process. Thus, although the primary platforms used in this PhD research are general purpose, rather than industrial-grade, the common principles are aimed to be universal.

2.5 Case system

In order to set the boundaries for the class of systems dealt with in this thesis, a common case system is considered throughout multiple included papers.

When it comes to vision-guided robotics, there exist different use cases in terms of joint behavior of a robot and a vision system. For instance, in many custom research testbeds, images from a camera and their processing results are treated as a sampled signal, which is continuously monitored, with image features used for direct control of robot joints. The architecture of `image_raw` publish/subscribe topics in ROS is particularly suited for such applications. In contrast, this thesis considers robot-vision interaction in a discrete event manner, i.e. when a part of robot logic is predetermined, and at specific events the robot control program requests data from the vision system. Such interaction is common to industrial robotic applications, as well as in manufacturing-related robotics research.

The case system used in this thesis is comprised of four components (Figure 2.3). The master control node sends commands to the robot server and receives the corresponding acknowledging responses. On certain events, it initiates a request to the vision system, which is handled by the image acquisition service. The latter hands the resulting images from one or more cameras to the image processing node, which, upon completion of processing, sends the results back to the master control node.

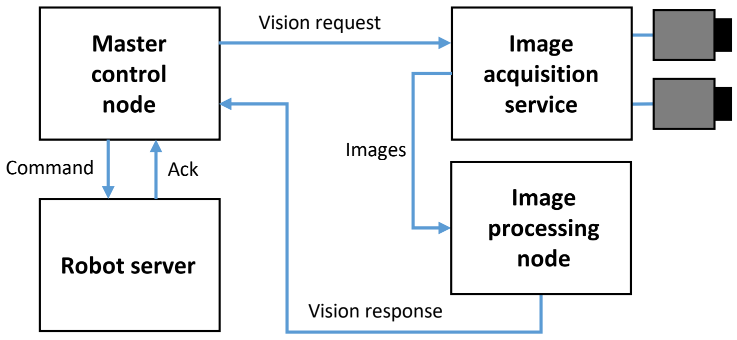


Figure 2.3: The case system high-level data flow principles

2.6 PhD research path

Prior to starting this PhD project, the author completed his master thesis [16] while working at SINTEF Raufoss Manufacturing AS (SRM) and being involved in research activities concerning combination of industrial robotics and machine vision. An interesting part of research practice was practical development of functional prototypes in the laboratory. The general impression was that the development process constitutes a form of craft, with the challenge to *prototype and integrate* multiple heterogeneous components: a robot control program made natively in the robot controller, a vision system implemented in a proprietary GUI-based vision software, Python scripts that communicate over RS-232 with part feeding equipment, Python scripts utilizing EtherCAT communication with various industrial equipment, and other components.

Thus, the initial motivation was to tackle the problematics of such early stage robotic systems development as done at SRM. During the master thesis work, a core focus of research was the calibration of robot/vision systems, specifically camera, stereo, and hand-eye calibration. These topics continued to be of high interest at SRM, so the goal was to continue working on calibration research as a part of the PhD project. The core idea in that respect was to explicitly capture and harness uncertainty regarding calibration parameters. A paper was published [4] with the initial results in this direction, but it was not included in this thesis to keep the scope of the thesis more coherent.

One of the most important ideas in this PhD project has been the notion of *vision pipelines*. When developing a vision algorithm, the developer thinks in terms of data flowing from one operation to another, with all of them forming a directed graph architecture. In most practical cases, however, such pipelines are just conceptual, with actual logic being implemented as a single function, a cascade of function calls, or configuration in a GUI-based vision software. When an algorithm gets too complex, with larger number of configuration parameters, it becomes harder to manage experimentation with the algorithm and tuning of its parameters. It was of interest, therefore, to investigate the feasibility of integrating explicit pipeline construction into the workflow of vision algorithm development with OpenCV and Python data science stack. Another idea was to be able to expose *reactive pipelines*, i.e. vision algorithms, defined as direct acyclic graphs, that would react to events of new images arrival. All of the pipeline-related considerations led to the **RQ1**, aimed at coupling the perception function with event-driven system behavior.

Open source software, such as OpenCV, the Python data science stack, and SRM's in-house code, was an integral part of the work on robot/vision calibration, and continued to be such in the vision modules developed during the PhD project. At the same time, it was challenging to integrate custom vision software modules with the image acquisition function. The latter was either a part of commercial vision software, with limited access to raw image data, or available through a low-level SDK, with the need for implementing custom adapters. This motivated the **RQ2** regarding exposing the image acquisition function as a service.

In the early stages of the PhD project, there was an idea of investigating possibilities for using formal methods to model event-driven behavior in robot/vision systems under consideration. This thinking was inspired by models of computation in the study of cyber-physical systems [10], as well as by Supervisory Control Theory [17]. However, this made the potential scope of the PhD work to broad. It was also challenging on a philosophical level to combine the specification/verification practices with ad-hoc prototyping approaches. As a result of publishing another paper not included in this thesis [5], the author received feedback from fellow computer vision researchers who deemed the idea of formal methods integration too cumbersome.

Nevertheless, it was still of interest to keep focus on certain system engineering principles. One of them is the abovementioned graph-based models of vision algorithms. Another principle was that of *composability*, i.e. the ability to automatically compose more complex logic from simpler parts. This was specifically interesting in the context of robot control, and motivated the **RQ3**. When working with the Adept Viper s850 robot, the employed architecture was based on a TCP/IP server in the robot controller that accepts commands from the outside and executes them. Such an approach is certainly not the best in terms of efficiency, as compared to custom code directly in the robot controller. However, it allows for extensive experimentation with high-level logic of the component that initiates the commands.

2.7 Thesis structure and contributions of the papers

The following chapters (3 – 7) in part I are aimed at setting a context for the methods and results described in the appended papers. The structure of the thesis follows the framework depicted in Figure 2.4, where the papers (the right column) create a logical sequence of ideas, and the chapters (the left column) highlight important theoretical underpinnings and overview the established practices. Each chapter logically precedes the corresponding paper, and the reader can refer to Figure 2.4 for a better understanding of the presented material.

An overview of problems and contributions in the included papers is presented below.

P1. Towards increased intelligence and automatic improvement in industrial vision systems

Problem

The case company, Kongsberg Automotive AS, is a heavy adopter of automation equipment with a high number of integrated vision systems. The latter are used for parts inspection and robot guidance. Through the course of manufacturing systems' operation, KA has been facing a range of challenges with robustness of vision sensing, mainly due to appearance variability of the imaged components. Before devising an improvement strategy for the available system, as well as for the planned new

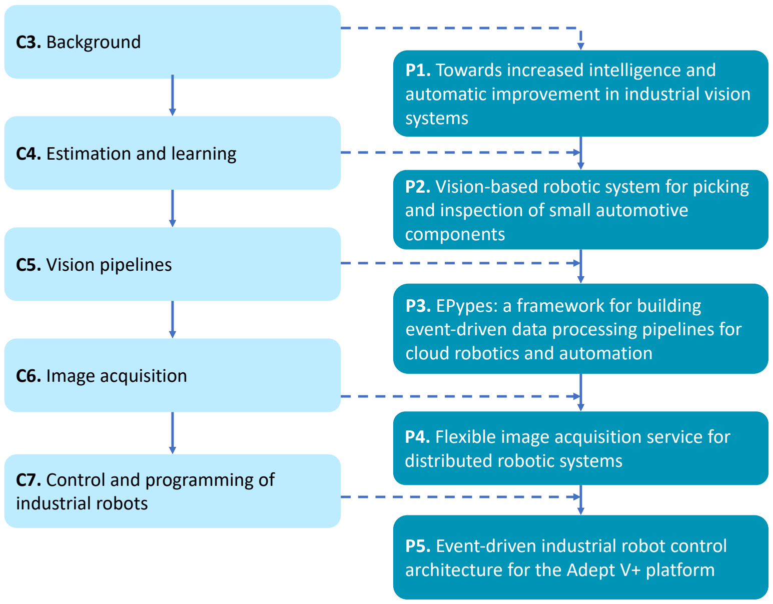

C3. Background
C4. Estimation and learning
C5. Vision pipelines
C6. Image acquisition
C7. Control and programming of industrial robots
P1. Towards increased intelligence and automatic improvement in industrial vision systems
P2. Vision-based robotic system for picking and inspection of small automotive components
P3. EPypes: a framework for building event-driven data processing pipelines for cloud robotics and automation
P4. Flexible image acquisition service for distributed robotic systems
P5. Event-driven industrial robot control architecture for the Adept V+ platform

Figure 2.4: Thesis structure

installations, there was a need for a systematic study of vision system functions, techniques and capabilities.

Contributions

- Case study of the company with focus on the KArtridge^{TM} product family and its star washer component.
- Study of the current state of the production systems at the KA facility in Raufoss, Norway, highlighting the vision systems capabilities.
- Formulation of the improvement strategy based on establishing a data store and data analysis system for on-line stream processing, off-line machine learning from historical data, and systematic adjustment of machine vision parameters.

P2. Vision-based robotic system for picking and inspection of small automotive components

Problem

The first paper was motivated by the need for better vision-based quality inspection of star washers, small parts used to manufacture automotive components by Kongsberg

Automotive AS. To eventually design a custom inspection station, the first steps in system development were to be realized with a flexible robot-based system with computer vision sensing.

The first apparent challenge was to design a vision algorithm to analyze star washers' appearance when lying on the feeder surface in order to determine the components' orientation for robotic picking. A related problem was to devise a vision algorithm for the analysis of a close-up image to check for the part's quality.

Contributions

- High-level planning of the prospective robot cell.
- Custom feature engineering algorithm utilizing the nature of a star washer as a circular object imaged from the top.
- Training a range of machine learning classifiers on the common set of feature vectors, with the aim of determining classifiers with the best performance.

P3. EPypes: a framework for building event-driven data processing pipelines

Problem

The motivation behind EPypes has been at the heart of this thesis work since its inception. When a new robotic system with vision sensing is developed, the early-stage system prototyping favors flexible tools and techniques that allow for iterating towards a functional solution quickly. A resulting prototype, however, is often developed as a patchwork solution, which may be challenging to effortlessly evolve into a production system.

The high level goal of the paper is to devise a system development method, backed by a set of tools, that would bring more structure into the prototyping phase without compromising the flexibility of the established data science/computer vision stack. Since the target area of the paper constitutes robotic systems with vision sensing, the developed prototypes should be easily integrated as part of distributed systems, possessing communication capabilities and event-driven behavior.

Contributions

- EPypes, a Python-based framework for building event-driven data processing systems that directly harness the data flow nature of vision algorithms, as well as asynchronous publish/subscribe messaging systems.

- A lightweight Python module for specification, scheduling and visualization of executable directed acyclic graphs (computational graphs).
- A set of abstractions that transform a computational graph into a reactive data processing pipeline with a generic queue-based input/output.
- Custom reactivity components based on ZeroMQ publish/subscribe mechanisms.
- A system development methodology based on the proposed tools, which proceeds from the early stages of algorithm development to a structured distributed system with well-defined capabilities and behavior.
- Validation of the proposed tools in a lab-based distributed computing environment, featuring ZeroMQ-based publish/subscribe communication, with quantitative assessment of timing characteristics.

P4. Flexible Image Acquisition Service for Distributed Robotic Systems

Problem

Cameras based on the GigE Vision standard are pervasively used as a part of industrial automation systems. Nevertheless, they are traditionally used either with off-the-shelf vision software with limited flexibility, or as directly embedded in custom software modules. Some of the cameras are supported in ROS, but mainly as providers of `image_raw` topics. With the proliferation of more distributed setups and flexible robotic architectures, the workflow of image acquisition needs to support a wider variety of communication styles and application scenarios.

In a nutshell, this paper aimed at proposing a generic image service supporting continuous acquisition from several cameras internally, with flexible service-oriented capabilities for external systems. The described flexible image service had been a missing link in the robot-vision architecture that underlies this thesis.

Contributions

- Multi-threaded architecture of the vision service.
- Collection of concurrent data structures for short-period image caching, search and retrieval via a time stamp.
- A driver for Allied Vision cameras with GigE Vision interface based on the vendor's Vimba library.

- A high-level Python extension for the core C++ code.
- Quantitative evaluation of timing properties of the vision service.

P5. Event-driven industrial robot control architecture for the Adept V+ platform

Problem

The initial planning of robotic cell in paper 2, as well as corresponding experiment with physical equipment, involved an Adept Viper s850 robot. Its controller runs V+ real-time operating system, and can be programmed in the V+ language. Although developing a robot program directly in the controller is a common practice, the resulting solutions are rigid and difficult to integrate into a distributed automation system. It is also hard to quickly iterate towards the final solution.

To create a more flexible control module for an Adept robot, this paper proposes an architecture based on the robot controller providing a TCP/IP server and a collection of robot skills, and a high-level control module deployed to a dedicated computing device. The control module possesses bidirectional communication with the robot controller and publish/subscribe messaging with external systems. A particular control module is composed of a set of concurrent coroutines, which allows to implement complex logic in a well-defined way.

Contributions

- Multi-tasked server realized in the robot controller with the ability to accept commands from a client (robot control node) and launch the corresponding robot skills.
- A Python library, based on the AsyncIO coroutines, for building custom robot control nodes deployed to a dedicated computer; the coroutines can be flexibly composed together to realize event-driven robot control logic and communication with external systems.
- An application-level protocol for communication between a robot server and a master control node.
- A network architecture based on virtual LANs for connecting the robot network and the vision network.
- A vision algorithm, defined as an EPypes computational graph, for measurement of sharpness of a calibration object attached to the robot's tool plate.

- Validation of the proposed solutions on a robot application with visual feedback, realized as a distributed system, for automatic determination of the robot pose where the robot's tool plate appears most in focus in the close-range camera.

Chapter 3

Background

This chapters provides an introduction to the application area context of the work underlying this thesis, namely machine vision for industrial automation. At first, the specifics of the case company, Kongsberg Automotive AS, are introduced, highlighting the problematics around vision systems for imaging small parts. Further, the overview of automation in manufacturing and industrial vision systems is presented. An emerging area of cyber-physical systems and their application for manufacturing is then described. The chapter is concluded with an overview of concurrency and parallelism.

3.1 Specifics of Kongsberg Automotive AS

Kongsberg Automotive AS is a global manufacturing company producing components and subsystems for the automotive industry. This thesis is concerned with the production facilities of the KA plant in Raufoss, Norway, which supplies products for vehicular fluid transfer, marketed as Raufoss ABC$^{\mathrm{TM}}$. Raufoss ABC$^{\mathrm{TM}}$ is a product system providing a function of coupling air brake tubes and targeting the commercial vehicle market (buses and trucks). Raufoss ABC$^{\mathrm{TM}}$ includes air brake couplings, building blocks, release tools, and rotation stops.

KArtridge$^{\mathrm{TM}}$ is a product family of composite couplings with a metallic star washer and clamp ring, and a series of rubber O-ring seals. A coupling from the KArtridge$^{\mathrm{TM}}$ family is shown in Figure 3.1. It consists of a housing, the inner parts (cone element, environment seal, clamp ring, seal tube, and support sleeve), and the outer parts (main port seal, locking ring, star washer, and environmental port seal).

Most of the assembly operations at KA are performed by dedicated transfer machines, optimized for performance and fulfilling the requirement for high volume pro-

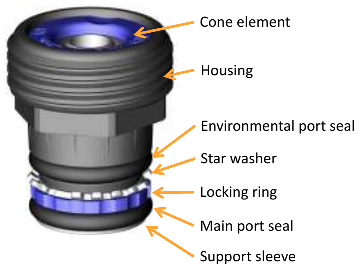


Figure 3.1: A KArtridge^{TM} coupling

duction. Each machine is designed by a specific machine supplier, and is typically comprised of the supplier's standard modules.

KA utilizes a high number of vision systems installed at various production stages and serving the functions of individual components inspection, process inspection, and object pose identification for picking. The installed machine vision solutions are off-the-shelf, and heavily rely on the associated illumination setups. The latter are implemented using LED solutions of various sizes and colors, and are controlled to turn on coinciding with part arrival and camera exposure.

The question regarding the role of illumination in vision systems is of major importance for KA. A general consideration is on the question of what types of lighting are better for different materials. KA deals with components made of brass, composite (having different colors), and rubber. Especially difficult is the problem of O-rings recognition because of process-caused color irregularities: O-rings' surface color can vary from black to gray, and the application of silicone oil induces reflection.

Some of the assembly lines include robotic manipulators and flexible feeders. In the latter case, the parts to be picked are randomly distributed on the feeder surface, and vision systems in combination with robots (typically gantry type) are used to pick the parts.

The problem of pose estimation using a mono vision system is challenging for relatively big parts that have a large degree of freedom in terms of orientations. In the KArtridge^{TM} assembly, the housing and the cone element are characterized by such geometry.

Because there exists time-dependent variability in the dimensions of the parts (specifically those obtained from the external suppliers), KA is interested in more effective quality process control with application of vision systems.

A star washer plays a critical role in the KArtridge^{TM} assembly by securing the grip function between a coupling and its housing. It is important that it is assembled in the correct orientation, so that the teeth will create resistance against the housing after assembly. In addition, the washer's teeth need to be of the required geometry. An example of a good and a defective star washer is presented in Figure 3.2. The failure conditions in this case constitute wrong geometry of some of the teeth.

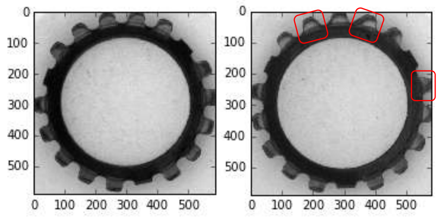
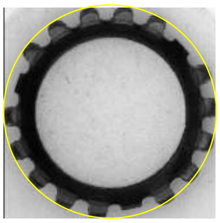


Figure 3.2: An example a good star washer (left) and the one with various teeth geometry defects (right)

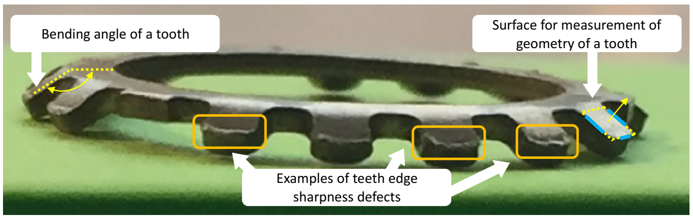
Depending on the product size, the outer diameters of star washers range from 15 to 26.5 mm. In addition to the small dimension of the star washers, their teeth, which perform the most important function, are much smaller, ranging in width from 1.4 to 2.11 mm.

Vision-based quality control of star washers is an important task for KA. The geometric requirements, see Figure 3.3, include the outer diameter, geometry of each tooth, bending angle of each tooth, and edge sharpness of each tooth. As shown in Figure 3.3a, the outer diameter of a component is well-imaged from the top-down perspective. However, other characteristics require a more intricate setup. Methods for analysis of top-down images of star washers are described in paper 2 and applied to the problem of ML-based classification of star washer orientation when lying on the surface of a feeder.

Figure 3.3: Geometric features related to star washer quality



(a) View of a star washer from above, highlighting the outer diameter


Bending angle of a tooth
Surface for measurement of geometry of a tooth
Examples of teeth edge sharpness defects

(b) View of a star washer from the side, highlighting examples of degraded edge sharpness and the area of interest for a tooth geometry measurement

The challenges in developing a vision system for inspection of star washers include (1) the reflective surface of the parts, (2) batch-to-batch color variation, and (3) small dimension of the parts and the teeth respectively.

3.2 Automation in manufacturing and assembly

Automation is an integral part of the contemporary manufacturing systems. It is realized by technological solutions, including mechatronic equipment, control systems and programs of instructions, allowing to accomplish a process with limited human assistance, increasing productivity, and leading to higher product quality [18, 19]. Typical application of automated solutions in manufacturing target operations that are dull, dangerous and error-prone, and encompass such areas of activities as formative/subtractive/additive processes, material handling and movement, inspection, assembly, and packaging [19].

Industrial automation holds strong ties with the field of control engineering. However, because of the discrete nature of processes in manufacturing, control systems that govern them to a large extent resemble discrete programs of instructions. The latter are typically implemented as *sequential control systems* using programmable logic controllers (PLCs). For each work cycle, producing a part or a number of parts, the associated control system automates the processing steps comprising this work cycle.

In a simple case, the instructions are predetermined, whereas in a complex case, decision-making is included to provide response to such variations as operator interaction, different product styles, and inconsistency in the starting work units [18]. This category of applications require integrating sensor technology, particularly vision systems.

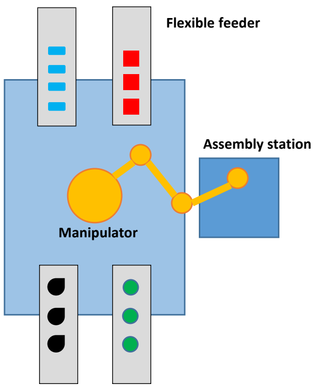
Assembly constitute a vital part of modern manufacturing systems, and is concerned with producing compound products from individual parts and sub-assemblies. Because many assembly processes require a high level of dexterity, they are often performed manually. However, because of requirements in higher quality, speed and repeatability, automated assembly is becoming more pervasive in manufacturing companies.

The main operations in assembly processes are parts mating, parts joining, parts recognition (position and orientation of randomly-fed parts), inspection, and material handling [20]. The latter has a special role in automated assembly, and deals with feeding parts into the system, handling of palettes, fixtures and tools, removal of completed products from the system, as well as transportation related to rework

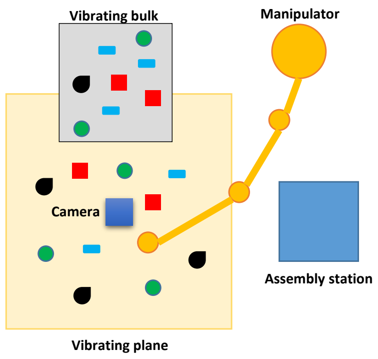
[21, 22].

As important objectives in feeder design/selection are their speed and reliability, it is a common practice in assembly automation to utilize part feeders mechanically tailored to parts with a specific size and shape. These include such systems as linear vibrators, vibratory bowl feeders, and belt feeders [21].

Flexible assembly systems typically make use of a number of dedicated feeders, each tailored to a specific part type. An alternative approach is a *fully-flexible assembly system* (F-FAS) concept, presented by Rosati et al. [23] and Finetto et al. [24]. The idea of F-FAS is to simplify the mechanical structure of an assembly system through a heavier usage of vision systems for parts identification and measurements. A typical solution of this kind comprises a flexible feeder handling a range of different components, a robotic manipulator, and an assembly station. A graphical comparison between a general assembly system and an F-FAS is presented in Figure 3.4.



(a) FAS



(b) F-FAS

Figure 3.4: Comparision of FAS and F-FAS

Though this solution may be unsuitable for high-speed production, its flexibility can allow for such application as flexible small-batch production, 100% inspection of parts in a measurement station, and flexible feeding to high-speed production lines. In the work underlying this thesis, the ideas of F-FAS were used for the proposed laboratory-based robotic inspection cell (see paper 2).

Part feeding in the laboratory work underlying paper 2 is done using an Anyfeed SX240 flexible feeder (Figure 3.5). It is designed for random feeding of parts and usage in a conjunction with a vision system for pose estimation. The picking surface of the feeder is backlighted, which simplifies further image processing and parts identification. A control computer can communicate with the feeder by means of

RS-232 communication interface to perform such operations as feeding more parts to the feeding surface, flipping the parts on the feeding surface, and moving the parts linearly. This is achieved by vibrations of different forms and intensities.

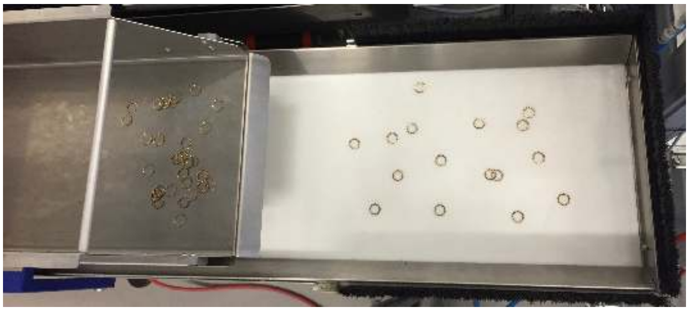


Figure 3.5: A flexible feeder with star washers

Assembly planning constitutes a high-level set of activities intended for mapping formalized assembly instruction to automated operations, e.g. those that can be performed by a robot. The assembly planning activities include CAD modeling of parts, tolerance modeling, workcell planning, sequence planning, mating pose determination and others [25].

When the results of assembly planning are mapped onto robot operational level, uncertainty becomes an inevitable part of the process, and sensory feedback serves the primary role of tackling it. Typically used types of sensors in assembly are force, torque, and tactile sensors, sensorized compliant devices, vision systems, optical sensors, mechanical probes, positional sensors, as well as sensors for measuring temperature, pressure, acoustic emissions, and acceleration [20].

3.3 Industrial vision systems

3.3.1 Overview

Change and uncertainty are inherent in flexible and reconfigurable manufacturing systems. In order to handle uncertainty in geometric shapes of the parts, as well as to allow flexible robot operation in geometric volume, vision-based sensing is typically applied [26].

Computer vision is an engineering discipline dealing with extracting useful information from images [27]. A computer vision algorithm takes an image, a set of images, or a video as input, and produces the application-specific output. *Machine vision* is branch of engineering focusing not solely on the individual vision algorithms, but on integration of all the technical components (hardware and software), needed for development and deployment of vision systems in the industrial context [28, 29].

The application domains of machine vision include the following [28]:

1. Defect detection: determining product defects, differentiating between different types of defects, including acceptable and unacceptable;

2. Guidance and alignment: providing a robot control program with visual estimate of an object pose or geometric displacement;

3. Measurement: deriving metric estimates of geometric features of a physical object;

4. Assembly verification: determining the correctness of an assembly process.

The inspected features of industrial product or process that are measured by vision systems include dimensional quality, structural quality, surface quality, and operational quality [28].

A vision system is typically comprised of such components as camera sensors with the attached lenses, illumination equipment, image acquisition technology, cabling, computer systems, and software [30]:

Any vision system is consisting of the image capture/acquisition part, responsible for obtaining the original image data and transforming it to memory, and computational part, aimed at processing the data to extract the desired information. Both parts introduce a number of requirements in order for the resulting vision system to be effective. During image capture, factors such illumination, quality of cameras and lenses play an important role in obtaining image data with high quality. When it comes to image processing and feature extraction, even more challenges arise: noise, shadows and reflection require sophisticated algorithms to extract the information of interest robustly and precisely enough. The latter, in turn, are constrained by the available computational resources and processing time requirements. The acquisition process depends of the used communication medium (e.g. Gigabit Ethernet, USB3, Camera Link), with the specific performance characteristics of bandwidth, transmission speed, channel reliability, spatial constraints, system integrability etc.

On-line vision systems, which constitute a part of the production process, provide the necessary information (e.g. pass/fail classification or robot movement coordinates) at the cycle time of the process. Conversely, *off-line* vision systems are used for recording information and further analysis [30].

2D vision systems utilize images from a single camera, whereas 3D vision systems reconstruct a 3D scene as a point cloud, and are subdivided into passive and active. Passive systems perform reconstruction using shape-from-shading, shape-from-motion, or passive stereo vision. Active systems, conversely, project a pattern of light onto the scene and further detect its position Active systems are categorized by the underlying physical principles of point cloud generation, namely triangulation, time-of-flight or laser pulse, and interferometry [26].

Noise is a factor that is always attributed to vision-based measurements. However, analysis of images of small parts are especially sensitive to the presence of noise and lighting irregularities, due to relatively small amount of pixels, on which the part is projected. Thus, high-resolution imaging is very important for small parts. When doing close-range imaging using a camera with a lens having a big focal length, correct focusing is important. Paper 5 considers an application scenario of automatic alignment of robot end effector in front of the camera, so that it appears in focus.

3.3.2 Lighting

Vision measurement in the industrial context is typically done under controlled conditions with the appropriate lighting and low noise [28]. Lighting is an integral part of an imaging setup due to the inherent physical principles of light reflection: what a person (or a camera) sees is electromagnetic radiation reflected from objects in the field of view.

Controlled illumination is organized using one of the following techniques, or their combination (see also Figure 3.6):

Backlighting Highlights the background so that silhouettes of dark objects are clearly visible.

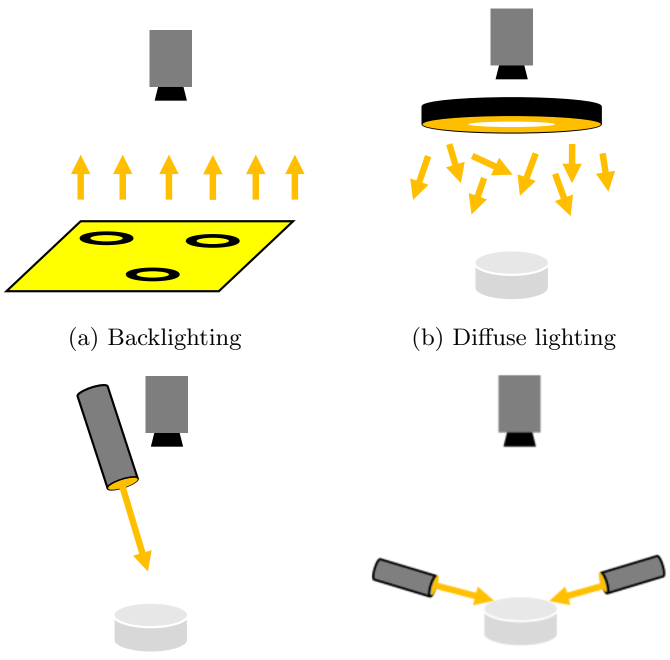
Diffuse lighting Spreads light rays in different directions to suppress specular highlight and make the image object appear more uniform.

Directional lighting Directs light rays towards the imaged object to highlight regions with reflection.

Dark field lighting Illumination from oblique angles to highlight non-flat regions on the imaged object.

3.3.3 Challenges

There exist numerous factors that influence the accuracy and performance of vision algorithms, including the measured object characteristics (size, shape, color,



(a) Backlighting (b) Diffuse lighting

(c) Directional lighting (d) Dark field lighting

Figure 3.6: Illumination techniques

texture), camera characteristics (camera resolution, quality of lenses), environment characteristics (pose, working volume, illumination).

Vision algorithms for industrial applications are typically very sensitive to the environment and the appearance of the observed objects. The measured part characteristics, such as color and reflectivity, may vary from one part to another. Lighting conditions are also difficult to maintain consistent [26]. Therefore, vision systems need to be robust enough to tackle this variability.

When camera measurements need to be expressed in real-world coordinates (e.g. for robot guidance or high-accuracy measurement), the quality of system calibration is of vital importance. In complex automated systems it is important to effectively integrate one or more vision systems with other components. In multi-sensor and multi-device environments (e.g. comprising cameras, lasers, and robots), one is typically concerned with integrating data from multiple sources under the same coordinate system [26]. These tasks require application of geometric computer vision methods [31].

Processing time is an important factor, particularly to industrial applications of vision systems [26, 28]. Because visual data typically has high-resolution, it requires efficient algorithms, often in combination with specialized hardware, to be processed. The processing time requirement is specifically critical for 3D imaging.

Flexibility in manufacturing should be supported by the flexibility of vision systems [28]. The abovementioned issues of part appearance variability and difficulty in establishing a consistent imaging environment naturally present a challenge to a greater flexibility. In addition, the organization of vision algorithms and availability of flexible and comprehensive computer vision libraries play a role in making vision-based solutions more adaptable. This aspect of vision systems development has received a special attention in this thesis, mainly with EPypes (paper 3) and FxIS (paper 4).

3.3.4 Perception and action

The key part of a vision system is its processing algorithm. It is no surprise then that algorithm development constitutes the core task in engineering of a vision system. Selection and integration of hardware, such as camera sensors, lenses, lighting equipment, communication infrastructure, and processing hardware, is also important, especially if the goal is to establish a highly controlled imaging environment.

In practice, however, vision systems do not exist in isolation — they are integrated into larger systems comprising robots, assembly machines, material handling devices etc. Although, thanks to robust feature detection algorithms, there exist many ”consumer-grade” computer vision systems, in manufacturing one is normally concerned with development of custom control systems utilizing machine vision as a sensing modality. In such situations, two tasks, *perception* and *action*, need to be effectively integrated, and vision algorithms should not be considered in isolation (as solely procedures for processing images).

In automated systems, *perception* is the process of representing the sensory information in a task-oriented model of the world [32]. In artificial systems perception is realized by means of sensing and estimation. One distinguishes between two types of perception: *proprioception* recovers the internal state of the systems (e.g. a robot), while *exteroception* recovers the state of the external world. In the former case, passive sensors are typically used to acquire data, in the latter case — active (contact and non-contact) sensors [32].

This thesis concerns vision systems, which are the primary means of realization of exteroception in robotics. Naturally, biological vision has been an important source of inspiration for researchers dealing with artificial vision. Yet, an important characteristic of biological agents is an inherent interplay between perception and action, embodied in numerous behaviors reliant on sensory information. These include various body motions, as well as eyes getting focused on particular aspects of the scene. In addition, perception and action are greatly related to planning and other types of complex cognitive processing.

3.4 Cyber-physical systems

A *cyber-physical system* is defined as a composition of physical subsystems with computing and networking, where computing devices communicate with each other and interact with physical subsystems with the means of sensors and actuators in a feedback loop [9, 33]. The above definition has a lot in common with a classical notion of a control system. In fact, it was acknowledged by prominent researchers in control that the emergence of CPS can be seen as a natural consequence of the practical development of control systems [34].

Historically, control systems included two types of equipment, corresponding to continuous and discrete control respectively. The former included controllers, recorders, and displays, located on the control panel. The latter were implemented as relay cabinets for start and stop sequences, and safety interlocks, corresponding to the continuous and discrete control respectively [34]. The field of control engineering has been constantly adopting various computer technologies such as digital computers, microcontrollers, computer networks and various sorts of software. In the recent years, the role of network-based communication and software sophistication in control systems has increased. This in a way has bridged the continuous and the discrete world of control, by putting a particular emphasis on joint dynamics of real-world processes and computer-based control. This demanded a novel system science, the intellectual core of which would constitute models and abstractions conjoining computational and physical dynamics [9].

Thus, the interdisciplinary filed of cyber-physical systems has emerged [34], combining knowledge from control theory, concurrency theory, hybrid systems, formal methods for specification and verification, distributed algorithms, model-based design, and real-time systems [33]. Because systems that interact with the physical world are often implemented as embedded systems, the intellectual core of CPS has been formed within the embedded and real-time systems community.

A CPS example in presented in Figure 3.7. Computing part of a CPS may be realized with one or more *computing platforms*, each comprised of one or more computers, sensors and actuators. Computing platforms communicated with each other by the means of *network fabric*. The controlled physical processes form the physical plant, which can encompass mechanical, chemical, biological processes, as well as human operators. The *physical plant* is perceived by sensors and acted upon by actuators of the respective computing platforms [9].

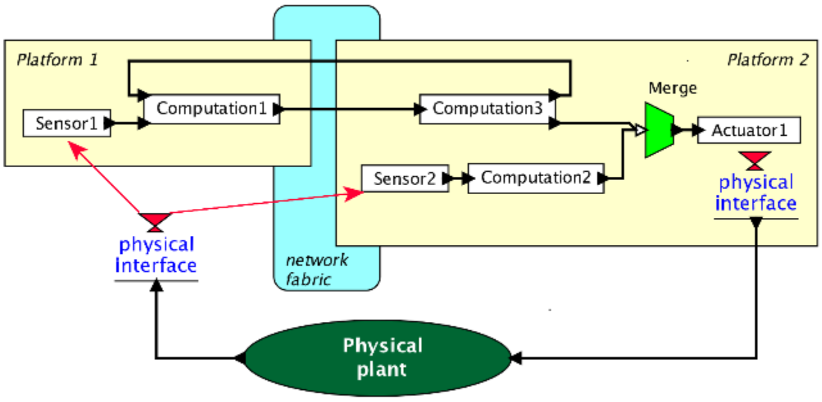


Figure 3.7: An example of cyber-physical system components by Lee and Seshia [9]

One may argue whether the CPS concept brings anything new to the field of control systems, or if it is just a buzz word encompassing the technologies that has already been around for a while. To confront this point of view, one may point out that cyber-physical systems shall be regarded as a multi-disciplinary field that puts a greater emphasis on the computational and networking subsystems. The distinguishable features of cyber-physical systems include reactive and real-time computation, inherent concurrency, feedback control, and safety-critical requirements [33]. The multi-disciplinary focus of CPS is expected to allow development of large scale control systems [34], tackling the issue of complexity, and introducing greater flexibility. This thesis has been greatly influenced with the CPS thinking, specifically with respect to the concurrency and communication aspects.

Computational systems that control physical processes and communicate with each other are already an integral part of advanced manufacturing systems. Therefore, the field of CPS has been of particular interest among the researchers in manufacturing [35, 36], where the term *cyber-physical production systems* has been coined.

In manufacturing, the focus has largely been on application-level approaches highlighted in strategic initiatives like Industry 4.0 [12], Industrial Internet [13], and Smart Manufacturing Systems [14]. In this case, numerous ideas have been drawn from earlier research undertakings in holonic and multi-agent control systems, service-oriented architecture for manufacturing control, manufacturing-related data mining, virtual manufacturing, wireless industrial communication, and human-centered research. Generally, the abstraction level of research in cyber-physical production systems is much higher than that of the original CPS, and has a distinctly application-specific focus.

Because future cyber-physical systems in the industrial automation settings are inherently heterogeneous, i.e. consisting of mechatronic devices based on different computational platforms and programmed in different programming languages, the middleware technology in combination with open architecture are seen as important prerequisites for their effective integration [35]. Middleware is considered in more details in chapter 7 and paper 5.

3.5 Concurrency and parallelism

Today's computing systems, especially those with distributed components, come with an inherent interplay between concurrency and parallelism. These are related, but conceptually different, computing problems.

Parallelism concerns computationally-intensive tasks, whose execution efficiency is constrained by the available processing power, and/or the available memory. Typical examples of this kind are computer vision algorithms, training of machine learning models, real-time signal processing, stream processing, robotic mapping, etc. To satisfy the respective timing requirements one typically strives to harness parallel computing capabilities, including multi-processing systems, GPUs, computing clusters, and cloud resources.

Concurrency is a conceptual notion of simultaneous occurrence of different processes. Comparing to parallelism, which concerns computations that physically execute simultaneously [9], concurrency is about *dealing with*, rather than *doing*, lots of things at once [37].

In computational systems, concurrency is expressed on some abstraction level (in programming language constructs or modeling formalisms). The problem of modeling concurrency arises in several contexts. This includes, but is not limited to, heterogeneous industrial automation systems, publish-subscribe middleware, servers supporting multiple connections at once, and interactive applications, where input from users constitutes sporadic events.

In distributed systems concurrency is tightly intervened with communication. The same holds true for cyber-physical systems, where concurrent models of computation [9] are extensively used to synthesize the desired system behavior. In practical software engineering in distributed systems, concurrency is typically associated with I/O-bound execution. To allow for multiple connections and deal with the waiting time, such techniques as asynchronous functions calls, futures, event loops, pools of workers etc are used. These techniques, which explicitly acknowledge concurrent execution, can also be applied to tackle CPU-bound problems and harness the parallelization capabilities.

In this thesis, concurrency is harnessed in the following ways:

- Thread-based concurrency with synchronization over blocking queues in Python is used in the design of EPypes (paper 3).
- Thread-based concurrency in C++ is used in FxIS (paper 4) to realize image acquisition and service request/response. Synchronization and data management is realized with custom concurrent data structures.
- Publish-subscribe architecture based on ZeroMQ for connecting the robot control node, the image acquisition service, and the image processing node (used in both paper 3 and paper 5).
- Cooperative multitasking using Python's AsyncIO coroutines is used in `pyadept` (paper 5) to achieve composition of communication-heavy robot logic.

Chapter 4

Estimation and learning

Estimation can be loosely defined as "the computation of some transformation or other mathematical quantity, based on measurements of some nature" [31]. Estimation of unknown parameters and functions from noisy data is a pervasive task in many fields of engineering. Automation and robotic systems always utilize some sensing modalities, and the increased availability of sensors of various kind offers better possibilities in terms of estimating internal state of the systems and perceiving the environment around. *Learning* is a related task, with a broader meaning, associated with the ability of a system to improve its function as a result of observation of some kind. The methods by which this improvement is achieved are very often the same as in the classical estimation, i.e. are based on various models with statistical assumptions, cost functions, and optimization methods. Nevertheless, the fields of artificial intelligence and machine learning bring additional approaches of their own.

There exists a large body of knowledge on estimation methods, the associated models, optimization algorithms, and the underlying probabilistic assumptions. The approaches developed so far share the common principles, but, depending on the application area or the scientific school, have different taxonomy, notation, and even philosophy behind them. Examples are general-purpose model fitting, sensor calibration, estimation of probability distributions' parameters, machine learning, Bayesian estimation and so on. Some scientific schools put an emphasis on the probabilistic nature of the estimation process, others concentrate on efficient implementation, typically harnessing the capabilities of linear algebra, while some attempt to tackle the problems algorithmically by employing non-trivial models, cost functions, and data structures.

Recent years has been characterized by an increased interest in the use of machine learning methods for realization of highly-complex data-driven systems. The whole new discipline is emerging, namely *data science*, which goes beyond the classical data analysis methods, and combines interdisciplinary knowledge in probability and statistics, machine learning, computer science, and practical software engineering to tackle complex problems with data. The latter in the data science context is often unstructured and requires a great deal of prototyping/hacking.

There are several ways to generalize estimation and learning problems: from the points of view of classical statistics, Bayesian statistics, linear algebra, or non-linear optimization. One could put emphasis on models, or on algorithms. Also, one could be tightly application-specific, or generalize on broader sets of similar problems. This chapter establishes a common vocabulary for estimation and learning, which is based on three integral components [38]: (1) model representation, (2) evaluation function, and (3) optimization algorithm.

4.1 Machine learning

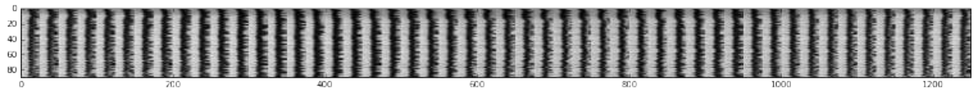
Machine learning (ML) is a field that studies computer algorithms for automatic learning (”to do something better in the future”) from observation data (”based on what was experienced in the past”). That is, an ML application is associated with a particular *task* that has to be improved by learning rather than by implementation of an imperative procedure [39, 40].

ML is typically considered a sub-area of artificial intelligence (AI), though it is highly related to statistics, optimization, computer vision, and other disciplines. There exist distinct scientific paradigms studying ML such as statistical learning theory and computational learning theory.

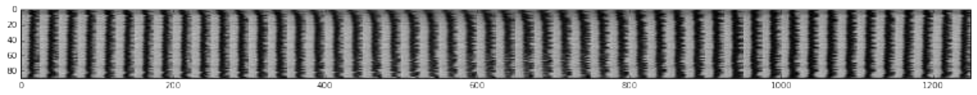
A large group of ML techniques, dubbed *supervised learning*, is focused on learning an unknown function $f_{\mathbf{p}} : X \to Y$ from a set of training samples $\{\mathbf{x}_i, \mathbf{y}_i\}$, where $\mathbf{x}_i \in X, \mathbf{y}_i \in Y$, and $\mathbf{y}_i = f_{\mathbf{p}}(\mathbf{x}_i)$. Hence, the known data samples are used to estimate the unknown function, which can be used for regression ($Y \subseteq \mathbb{R}^d$) or classification ($Y = \{0, 1\}$ or $Y = \{0, 1, 2, \ldots, k-1\}$, where k is the number of classes).

In paper 2, several supervised classification algorithms were tested on the same training data. The latter, formed using a custom feature engineering procedure from a set of images of star washers. Figure 4.1 shows two feature matrices, rendered as images, where each row corresponds to a single feature vector obtained from an individual star washer image.

Function $f_{\mathbf{p}}(\cdot)$ typically constitutes a complex model with a large number of parameters. For many problems the learned $f_{\mathbf{p}}(\cdot)$ would surpass possible imperative implementations, while for many, it would not be possible at all to hand-craft equiv-



(a) Feature matrix for $y_i = 1$



(b) Feature matrix for $y_i = 0$

Figure 4.1: Feature matrices for star washers' orientation classification

alent procedures.

Supervised ML algorithms include linear regression, logistic regression, artificial neural networks (ANN), support vector machines (SVM), decision trees, adaptive neuro-fuzzy inference system (ANFIS), and others.

Unsupervised learning algorithms are aimed at finding regularities in unlabeled data, i.e. in a set $\{\mathbf{x}_i, i = 1, ..., m\}$, without dependent variables $\{\mathbf{y}_i\}$. Thus, unsupervised learning algorithms are used for clustering, dimensionality reduction, anomaly detection, and often are applied for preprocessing the original data before training some of the supervised learning algorithms.

Depending on the application, one could aim at building ML models for prediction or for inference. In the former case, it is of interest to predict an unknown value $\mathbf{y}^* \in Y$ given an observation $\mathbf{x}^* \in X$. Conversely, in the latter case one aims at building an *interpretable* model that describes the nature of mapping $X \to Y$ [41].

ML is often used for analysis of image data, due to better accuracy of functions for recognition of complex patterns learned from data, as compared to hand-crafted procedures [40].

Many ML implementations greatly benefit from large data quantities and powerful computational infrastructure. Software frameworks as Apache Hadoop and Apache Spark provide capabilities of efficient machine learning from big data using *batch processing* with map-reduce technique. In some situations the data is not already stored, but arrives online. To apply machine learning in this case, *stream processing* engines, such as Apache Storm, are employed.

4.2 Models

Every estimation problem is coupled with one or more models, constituting a formalized description of a phenomenon at hand. A model $M_{\mathbf{p}}(\cdot)$ can closely mirror a physical system, e.g. model of a sensor, or have a custom intricate structure, such as models used in many machine learning applications. The former case is natural when one possesses knowledge or assumption of how a physical process can be mathematically described. In computer vision, the typical examples is knowledge on light transport and camera structure, which is formalized with the linear pinhole camera model and non-linear lens distortion model. These models are well-interpretable, and thus can be used for inference. Other tasks, conversely, can be treated as pattern recognition problems, in which a new datum is interpreted based on the prior experience without an explicit physical model [27]. Here, models like neural networks, decision trees, support vector machines, random forests etc. are applied, with a primary focus of prediction (recall section 4.1).

4.3 Point estimation

Let D be a set of all possible data points, and P – a set of all possible parameters that have to be estimated. Thus, an observed datum is represented as a vector $\mathbf{x} \in D$. Likewise, the unknown parameters, which can e.g. characterize a particular model $M_{\mathbf{p}}(\cdot)$, are represented as a vector $\mathbf{p} \in P$. If P is a vector space, the estimation problem is referred to as *point estimation*. A typical example when this is not the case is the problem of pose estimation, since Euclidean space $SE(3)$ is not a vector space.

4.4 Linear models

For many systems, linear models are highly preferred due to their simplicity and interpretability. Also, with matrix/vector representation one is able to harness vectorization capabilities of the computing devices. Singular value decomposition allows to obtain a closed-form solution to model estimation problems.

In many cases, particularly in computer vision, an estimation problem is formulated as a homogeneous system of linear equations:

$$\mathbf{A}\mathbf{x} = \mathbf{0} \tag{4.1}$$

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$. Typically, one considers overdetermined problems, where the dimensions of data exceeds the number of unknowns, i.e. $m > n$. In this case, (4.1) has an infinite number of solutions, with the optimal $\mathbf{x}^+$ minimizing the squared norm of $\mathbf{Ax}$:

$$\mathbf{x}^+ = \arg\min_{\mathbf{x}} ||\mathbf{Ax}||^2 \text{ s.t. } ||\mathbf{x}|| = 1 \tag{4.2}$$

$\mathbf{x}^+$ is obtained from singular value decomposition (SVD) of $\mathbf{A}$, that is, given $\mathbf{A} = \mathbf{UDV}^T$:

$$\mathbf{x}^+ = \mathbf{V}(:, -1) \tag{4.3}$$

The estimate obtained from a linear model does not account for many non-linear and noise factors that are inherent in most physical phenomena. As such, a non-linear optimization routine is normally conducted with the linear estimate as the first approximation.

4.5 Optimization

Optimization is central to both machine learning and geometric computer vision (as well as all other problems involving estimation), since this is the primary procedure of determining the sought parameters. An optimization problem is formulated in a way that model parameters and constraints are based on prior information or parameter limits, and the objective constitutes a measure of misfit or prediction error.

Parameters of linear models, described in section 4.4, can be estimated in closed form. Conversely, problems with non-linear cost functions require suitable algorithmic procedures. There is a range of well-developed methods of non-linear optimization [42, 43], all following a common formal framework.

Let $\mathbf{p} \in \mathbb{R}^n$ be a vector of unknown parameters, which are of interest to estimate. $\mathbf{p}$ usually characterizes a model $M_\mathbf{p}(\cdot)$, but, depending on the problem that uses an optimization method, can have an arbitrary meaning.

There is a vector function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ which maps the parameter vector $\mathbf{p}$ to an m-vector of values, representing a problem-specific measure of misfit. For instance, in model estimation problems, to fit model $y = M_\mathbf{p}(\mathbf{x})$ to data set $\{(\mathbf{x}_i, y_i), i = 1, ..., m\}$, the corresponding function $\mathbf{f}$ will map to a vector of differences between given y_i and outputs of $M_\mathbf{p}(\mathbf{x_i})$, i.e.:

$$\mathbf{f}(\mathbf{p}) = \begin{bmatrix} y_1 - M_\mathbf{p}(\mathbf{x_1}) \\ ... \\ y_m - M_\mathbf{p}(\mathbf{x_m}) \end{bmatrix} \tag{4.4}$$

A cost function for least-squares problems is defined as follows[1]:

$$C_{SS}(\mathbf{p}) = \frac{1}{2} \sum_{i=1}^{m} (f_i(\mathbf{p}))^2 = \frac{1}{2} \mathbf{f}(\mathbf{p})^T \mathbf{f}(\mathbf{p}) \tag{4.5}$$

[1]SS for *sum of squares*

The leading $\frac{1}{2}$ does not change the outcome of optimization, but simplifies analytical derivations. Other equivalent forms of least-squares cost function is mean squared error (MSE) and root mean squares (RMS), defined as follows:

$$C_{MSE}(\mathbf{p}) = \frac{1}{m}\sum_{i=1}^{m}(f_i(\mathbf{p}))^2 = \frac{1}{m}\mathbf{f}(\mathbf{p})^T\mathbf{f}(\mathbf{p}) \tag{4.6}$$

$$C_{RMS}(\mathbf{p}) = \sqrt{C_{MSE}(\mathbf{p})} \tag{4.7}$$

There exist numerous methods for non-linear optimization. All of them are iterative, i.e., given a cost function $C(\mathbf{p})$, and the initial estimate $\mathbf{p}_0$, they produce a series of vectors $\mathbf{p}_1, \mathbf{p}_2, ...,$, which is intended to converge to the optimum $\mathbf{p}^*$ [43]:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} C(\mathbf{p}) \tag{4.8}$$

Descent optimization methods are based on computing a gradient of cost function at each iteration step, i.e. for $\mathbf{p}_i$ (where $i = 0, 1, ...$), gradient $\nabla C \in \mathbb{R}^n$ is computed as follows:

$$\nabla C = \begin{bmatrix} \frac{\partial}{\partial p_1}C(\mathbf{p}_i) & \frac{\partial}{\partial p_2}C(\mathbf{p}_i) & ... & \frac{\partial}{\partial p_n}C(\mathbf{p}_i) \end{bmatrix} \tag{4.9}$$

4.6 Flexible machine learning models

As was noted in section 4.2, one can strive to train more interpretable models and use them for inference, or make use of less interpretable, but more flexible models, mainly targeting the prediction task. Linear models, as well as their non-linear counterparts such as generalized additive models possess higher interoperability. By this reason they have been widely used for statistical modeling and system identification involving physical phenomena.

In many machine learning settings, one is primarily interested in prediction rather than inference, with the sought input/output relationship possessing high degree of complexity. In those cases, more flexible models are applied. These include neural networks, tree-based models, support vector machines, and ensemble methods. The latter, such as bagging and boosting, are based on the idea of training multiple learning algorithms and taking their weighted votes to achieve greater predictive performance.

Paper 2 is concerned with solving the binary classification problem given the set of feature vectors extracted from the star washer images. Here, there is no need to build interpretable models, and, therefore, a set of flexible classifiers are evaluated.

A downside of flexible ML algorithms is *overfitting* — the optimization routine naturally tweaks the model parameters so that misfit with the training data becomes minimal, but, as a consequence, a larger number of predictions using unseen data are made incorrectly, i.e. the trained model fails to generalize to new samples. To tackle this issue and make the trained models "smoother", a common approach is to add a *regularization* factor to the cost function. Given the initial cost function $C(\mathbf{p})$, where $\mathbf{p} \in \mathbb{R}^n$ and p_1 corresponding to the bias term, the cost function with regularization is defined as follows:

$$C^{(reg)}(\mathbf{p}) = C(\mathbf{p}) + \lambda \sum_{i=2}^{n} p_i \tag{4.10}$$

where λ is the regularization factor.

Because $C^{(reg)}(\mathbf{p})$ is minimized, the term with regularization ($\lambda \sum_{i=2}^{n} p_i$) suppresses the values of $\{p_i, i = 2, ..., n\}$.

4.7 Probabilistic interpretation

So far, the treatment of estimation and learning methods has been from the linear algebra and optimization perspectives. These abstractions are algorithm-friendly and thus favored and well-understood by practitioners. In order to formally reason about uncertainty in estimation, the probabilistic models are used.

The statistical learning theory [41, 44] considers machine learning problems from the perspective of classical statistics, one of the core components of which is the *additive noise model.* In general, a noise model represents probability of obtaining a datum y given model parameters $\mathbf{p}$, i.e. $Pr(y|\mathbf{p})$. For a deterministic mapping $\mathbf{x} \mapsto M_{\mathbf{p}}(\mathbf{x})$, the corresponding additive noise model is:

$$y = M_{\mathbf{p}}(\mathbf{x}) + \epsilon \tag{4.11}$$

where ϵ is a random variable representing the additive noise. Typically, noise is modeled as normally distributed, i.e. $\epsilon \sim Norm(0, \sigma^2)$. In this case, it is valid to write as follows:

$$Pr(y|\mathbf{x}, \mathbf{p}) = Norm(M_{\mathbf{p}}(\mathbf{x}), \sigma^2) \tag{4.12}$$

Section 4.5 presented cost functions based on sum of squares of the outputs of a multi-dimensional function. From the statistical point of view, it is of interest to perform maximum likelihood estimation (MLE), which maximizes the probability of the observed data under the estimated model. For a statistical model $SM_{\mathbf{p}}(\cdot)$, parametrized by $\mathbf{p} \in \mathbb{R}^n$ and describing data of dimension d, given a set of m data points $\{\mathbf{x}_i \in \mathbb{R}^d, i = 1, ...m\}$, the maximum likelihood estimate of $\mathbf{p}$ is defined as follows:

$$\hat{\mathbf{p}}_{ML} = \arg\max_{\mathbf{p}} \prod_{i=1}^{m} Pr(\mathbf{x}_i|\mathbf{p}) = \arg\max_{\mathbf{p}} \sum_{i=1}^{m} \log Pr(\mathbf{x}_i|\mathbf{p}) \tag{4.13}$$

The most straightforward application of MLE is estimation of parameters of probability distributions. For example, to estimate normal distribution parameters from data points $\{\mathbf{x}_i \in \mathbb{R}, i = 1, ...m\}$, the following MLE expression holds:

$$\hat{\mu}, \hat{\sigma} = \arg\max_{\mu,\sigma} \prod_{i=1}^{N} Pr(x_i|\mu, \sigma) = \sum_{i=1}^{m} (\frac{(x_i - \mu)^2}{2\sigma^2} + \log \sigma) \tag{4.14}$$

In paper 3, MLE is used to estimate log-normal distribution given sets of duration data (see Figure 4.2).

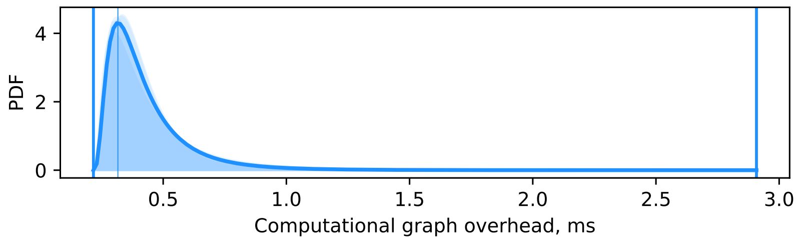


Figure 4.2: Log-normal PDFs estimated using MLE from paper 3

For model estimation problems characterized by a mapping $\mathbf{x} \mapsto M_{\mathbf{p}}(\mathbf{x})$ an additive noise model (4.11), and a data set of m pairs of independent and dependent variables $\{(\mathbf{x}_i, y_i), i = 1, ..., m\}$, the maximum likelihood estimate is the following:

$$\hat{\mathbf{p}}_{ML} = \prod_{i=1}^{m} Pr(y_i|\mathbf{x}_i, \mathbf{p}) = \sum_{i=1}^{m} \log Pr(y_i|\mathbf{x}_i, \mathbf{p}) \tag{4.15}$$

In case of normal noise, as in (4.12), the maximum likelihood estimate is equivalent to the least squares estimate.

4.8 Machine learning model evaluation

An intricate part of machine learning systems development lies in evaluation of different alternatives with respect to a set of performance metrics.

In order to train ML models without introducing data-related bias, the available data set is subdivided into three subsets (see Figure 4.3): training, validation, and testing sets, with the proportion of data being roughly 60%, 20%, and 20% respectively. Several alternatives are compared given the training and the validation set. The training set is used to estimate model parameters, which are in turn benchmarked with respect to the metrics of interest and the validation set. The performance of a fully-trained ML model is assessed in an unbiased way using the testing set.

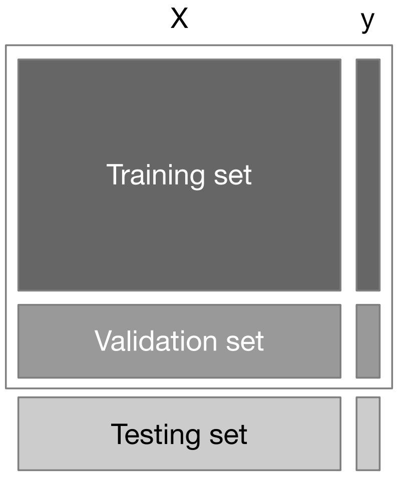
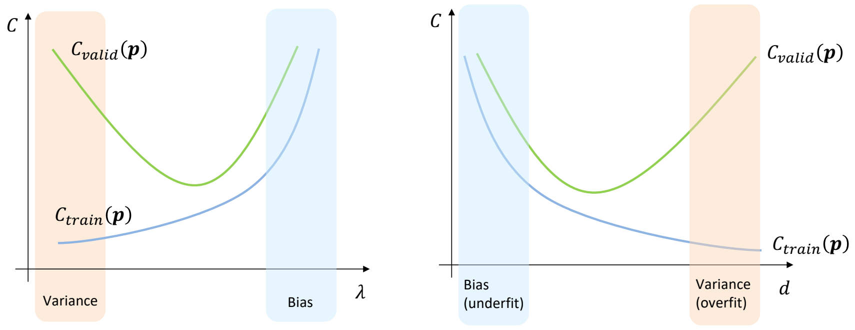


Figure 4.3: Training, validation, and testing sets

It is important to distinguish between *model parameters* and *hyperparameters* of a machine learning algorithm. A model $M_{\mathbf{p}}(\cdot)$ as a mathematical object is characterized by parameter vector $\mathbf{p}$, which is estimated using the training set. However, each learning algorithm possesses a number of parameters that don't characterize the corresponding model, but define the specifics of how the algorithm is executed. They are dubbed hyperparameters, and are tuned using the validation data set.

A commonly-used hyperparameter is a regularization factor λ (see section 4.6). If it is too small, the resulting model is risking to overfit data. In the opposite case, too simple model would introduce additional bias (Figure 4.4a). Similarly to the regularization factor, the dimensionality d of the model may lead to either high bias or high variance, and therefore needs to be tuned using the validation set (Figure 4.4b).



(a) Tuning a regularization parameter (b) Tuning dimensionality of a model

Figure 4.4: Model tuning to avoid high variance or high bias

With respect to classification problems, one can assess the performance of a particular classifier using the metrics based on the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) given the test set. The widely used metrics of this kinds are precision P, recall R, F-score F:

$$P = \frac{TP}{TP + FP} \tag{4.16}$$

$$R = \frac{TP}{TP + FN} \tag{4.17}$$

$$F = \frac{2PR}{P + R} \tag{4.18}$$

The above metrics are used in paper 2 to evaluate suitability of different supervised classifies to the problem of binary classification of star washer orientation. A summary of the evaluation is provided in Table 4.1.

Table 4.1: Comparison of classifiers from paper 2

Classifier	TP	TN	FP	FN	P	R	F
`SVC`	30	0	30	0	0.500	1.000	0.667
`MultinomialNB`	27	29	1	3	0.964	0.900	0.931
`GaussianNB`	29	30	0	1	1.000	0.967	0.983
`DecisionTreeClassifier`	27	24	6	3	0.818	0.900	0.857
`AdaBoostClassifier`	29	30	0	1	1.000	0.967	0.983
`RandomForestClassifier`	27	30	0	3	1.000	0.900	0.947
`KNeighborsClassifier`	24	29	1	6	0.960	0.800	0.873

Chapter 5

Vision pipelines

Computer vision algorithms that accept one or more images and produce the application-specific result are naturally modeled as data processing pipelines, i.e. describing signals/objects that get routed through a network of computational procedures.

The early stage of vision algorithm development is typically ad-hoc, with the focus on proving feasibility. For the sake of experimentation, limited structure is imposed on the code behind the prototype solution, so all the logic could, for instance, be composed as a single function. However, as the complexity of the developed prototype grows, it can be beneficial to evolve it as a structured entity. Otherwise, the resulting solution is likely to be difficult to maintain.

This chapter presents the idea of vision pipelines. The general pipeline pattern of typical vision algorithms is described first, followed by an overview of common image processing operations, specifically segmentation, connected components computation, edge detection, and feature detection/description. Further, the principles of computational graphs are presented, and two concrete examples of algorithms from this thesis, modeled as computational graphs, are discussed.

5.1 Images

From the perspective of applied mathematics, an image is considered a continuous function on D-dimensional domain Ω, mapping to C-dimensional vectors (representing values in each channel):

$$I : (\Omega \subset \mathbb{R}^D) \to \mathbb{R}^C$$

This continuous representations allows to reason about algorithms independently of domain discretization, e.g. by applying variational methods. In practice, however, the domain and range of an image are discretized. For a standard two-dimensional domain with width w, height h, and an `unsigned char` per pixel:

$$I_d : \Omega_{wh} \to \{0, 1, ..., 255\}^C, \text{where } \Omega_{wh} = \{0, h-1\} \times \{0, w-1\}$$

A binary image is obtained as a result of thresholding operator:

$$I_b : \Omega_{wh} \to \{0, 1\}$$

A label image represents a result of segmentation, mapping various regions $\Omega_i \subseteq \Omega_{wh}, i \in \{0, ..., k-1\}$ to the respective region indices:

$$I_l : \Omega_{wh} \to \{0, ..., k-1\}$$

5.2 Vision algorithms as pipelines

Any vision system can be structurally defined as a data processing pipeline. After image capture and acquisition, an input image is obtained in memory. It then follows a series of transformations, which leads to the application-specific output. An example of such vision pipeline is an abstract feature detection algorithm, comprised of the following steps:

1. Image enhancement (remove noise by smoothing, adjust brightness and contrast by gray value transformation);
2. Segmentation (thresholding, extraction of connected components, contour detection, edge detection, etc., and the combinations of thereof);
3. Feature detection (fitting of geometric primitives, template matching);
4. Decision making.

As an example, Figure 5.1 shows a sequence of operations that are common to many vision systems applications. In such general system, an image acquired by the camera is initially enhanced to simplify the later processing steps. After that, certain parts of the image are segmented, and the obtained parts are further used to detect the desired features.

Though the abovementioned sequence can be different, in has the same underlying principle. Generally, a vision pipeline proceeds ”from signals with almost no abstraction, to the highly abstract description needed for image understanding” [45].

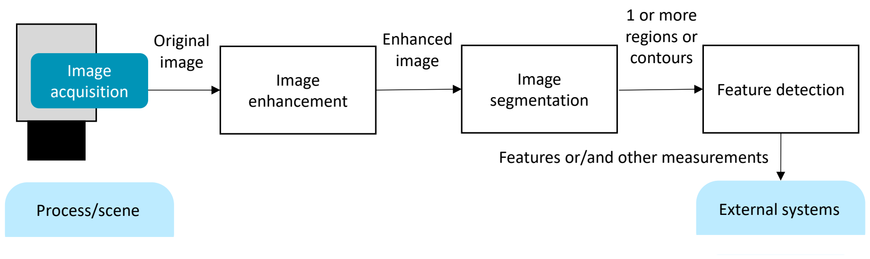


Figure 5.1: Common steps of a vision pipeline

Based on this idea, vision algorithm are often subdivided into two groups, depending on whether they are considered lower or higher level [45, 46].

The following sections provide an overview on the common image processing operations, specifically those used in this thesis.

5.3 Image segmentation

Image segmentation is concerned with identifying parts of the image that correspond to various aspects of the scene. A typical application aims at decomposing pixels belonging to background and foreground. In more complex situations one is interested in segmenting regions belonging to different objects. In some cases one is interested in partitioning image domain Ω into regions Ω_i:

$$\Omega = \bigcup_{i=1}^{n} \Omega_i \tag{5.1}$$

Other times, the additional boundary Γ, which separates the regions, is recovered:

$$\Omega = (\bigcup_{i=1}^{n} \Omega_i) \cup \Gamma \tag{5.2}$$

The simplest form of segmentation is thresholding – binarization of an image based on the given graylevel threshold value. This approach is the most computationally efficient, and works well for scenes with clear separation of background and foreground, specifically in highly-controlled machine vision setups. In this PhD project thresholding was used in situations where a dark object (star washer) was imaged against a bright background (paper 2), and for analysis of images of a custom calibration object on a white background (paper 5).

A downside of thresholding is in its reliance on a fixed graylevel value, which makes it not robust to scene appearance changes. To tackle this, several methods for automatic choice of threshold exits. The Otsu's method [47], for example, chooses the optimal threshold value by maximizing inter-class variance between the classes of white and black pixels. This methods was used in paper 2 as a part of the star washer segmentation pipeline (see also section 5.8.1 further in this chapter).

Thresholding performs a binary decision based solely on the local graylevel of a pixel. More elaborate segmentation methods take into account spatial context, which is more consistent with physics of image formation: if a particular pixel belongs to an object of interest, it is likely that its neighbors do as well. Such methods include the following:

- Clustering, maximum likelihood estimation of mixture models with the expectation maximization algorithms [48, 49];
- Region-based methods (based on building larger regions from smaller ones);
- Edge-based methods (label regions with the edges as boundaries);
- Variational methods (based on variational calculus; minimize cost functions dependent on an image) [50].

5.4 Connected components

In a undirected graph G, a connected component C constitutes a subgraph of G in which any two vertices are connected by a path in C, and no vertex in C is connected to any other vertex in $G \setminus C$.

A binary image can be thought of as a graph, with pixels as vertices, where there exists a connection between two vertices, if the corresponding pixels has intensity of 1 and are connected to each other. As such, computation of connected components given a binary image results in a label image and, optionally, a set of descriptive parameters for each component $i \in \{0, ..., k-1\}$, such as the component's center, top left corner of the bounding box, bounding box width/height etc.

Computation of connected components in a binary image was a central operation for segmentation of star washers in paper 2 and determining the region with the calibration object in paper 5. The latter use case is described in more details further in this chapter (section 5.8.2).

5.5 Edge detection

Edge detection is an image processing operation aimed at identifying groups of pixels where gray level across vertical and/or horizontal direction changes sharply. At the heart of edge detection algorithms lies maximization of image intensity gradients. Edge detection results in a binary image, where pixels with intensity 1 are labeled as belonging to edges.

Canny edge detector [51], which is the most widely used algorithm, in addition tracks edges by hysteresis and suppresses candidate pixels with weak connection to other edge pixels.

An edge image can be used to fit geometric primitives such as lines, circles, ellipses, and arbitrary-shape objects. A widely-used technique is the voting-based Hough transform [52].

5.6 Image features

A *feature* is a central notion in the majority of computer vision algorithms. Generally, it can be defined as a characteristic quantity extracted from an image [53]. There exists a number of types of features, differing by their dimensionality, sophistication of feature extraction algorithms, and the forthcoming applications. A feature is typically characterized by its location on the image plane, its quantitative description, or both.

One of the most trivial are region-based features. For a given region, one can compute area, center of gravity, moments (allowing to recover measures of the equivalent ellipse), convexity, parameters of smallest enclosing rectangle and smallest enclosing circle, contour length [53]. The same sets of features can be extracted from the subpixel-precise contour information.

Given a region and its original gray values, one can extract gray value features such as mean gray value withing the region, variance of gray values, α-quantile (used for robust contrast normalization), and gray value moments. The gray value may in some cases encode the degree of belonging to the region, allowing to make fuzzy decision and perform subpixel-precise measurements [53].

As mentioned before, another class of features are geometric primitives that are fit from an edge image.

The abovementioned features are largely used for gaging purposes. This is specifically useful for many machine vision applications, when quantifiable dimensionality information is of interest. However, in modern computer vision, particularly in multi-view imaging, features are primarily used for matching interest points in different views. The most basic in this respect are corners – points in image where two edges intersect. Corners are typical in images of man-made structures. They are also reliable to detect in calibration objects based on the chessboard pattern.

In the recent year, a range of robust feature extraction algorithms has been developed, all targeting the problem of reliable matching of interest points, and achieving invariance (full or partial) of scaling, orientation, affine distortion, and illumination. These algorithms include SIFT, SURF, HoG, GLOH, LESH, and ORB. The latter [54] is used in paper 3 for feature detection, description, and matching between two views in a stereo vision system.

5.7 Computational graphs

As motivated in Section 5.2, vision algorithms are conveniently modeled in a form of directed graphs. This section provides a short description of the computational graph abstraction from paper 3, used to directly specify executable vision pipelines. A computational graph constitutes a network of functions and data tokens, and is formally defined as a bipartite directed acyclic graph G:

$$G = (F, T, E)$$

where F is a set of functions, T is a set of tokens (representing data objects), and E is a set of directed edges between functions and tokens and vice-versa.

Tokens in T are subdivided into payload tokens and hyperparameter tokens. The latter are frozen on graph construction, and represent values that parametrize the algorithm. The former represent input data, as well as intermediate and final results.

An abstract example of a computational graph is shown on Figure 5.2. Rectangle vertices represent functions $F = \{f_1, f_2, f_3\}$, white circular vertices represent payload tokens, and gray circular vertices — hyperparameter tokens.

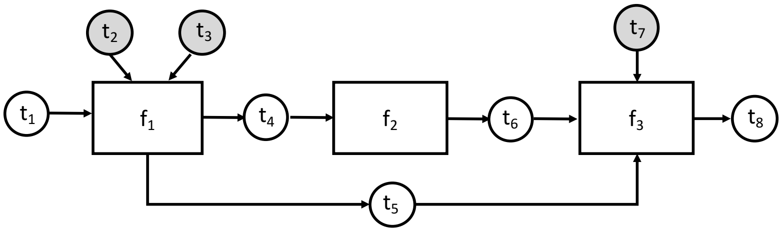


Figure 5.2: An example computational graph

Topological sort of G results in an order of vertices (functions and tokens) so that all the directed edges point from a vertex earlier in the order to a vertex later in the order. This is a standard approach to task scheduling, as it establishes an order in which all the precedence constraints would be satisfied. For the example in

Figure 5.2, the topological order for functions constitutes $f_1 \rightarrow f_2 \rightarrow f_3$. Invoking the functions in this order automatically executes the computational graph.

5.8 Concrete computational graphs

5.8.1 Star washer segmentation and polar transformation

In paper 2, an algorithm for star washer image processing is presented, in which the target object, imaged from the top-down perspective, is segmented, and further transformed to the polar coordinate system. The center of the latter should coincide with the center of the star washer. The result of the algorithm is further used for classification and inspection tasks. An improved version of the algorithm, formulated as an EPypes computational graph, is presented in Figure 5.3.

The original image is supplied to the Otsu's method to determine the optimal threshold value. The latter is used to perform thresholding, resulting in two binary images, corresponding to the object and the background respectively. The object thresholding (`im_t_object`) is supplied to the connected component analysis routine, which returns a label image and connected components statistics, packaged as a Pandas dataframe (`ccomp_df_object`). The latter is filtered using two hyperparameters describing the limiting sizes of the acceptable connected component. Because only one star washer is imaged, the result of filtering contains a reduced dataframe of size one. The location and bounding box dimension of this region of interest are used to crop sub-images from the original image (`image`) and the binary image of the background thresholding (`im_t_bg`). Sub-image of the latter is used for another connected component computation, with the goal of segmenting the background circle, appearing because of the hole in a star washer. The circle is filtered out (`select_bg_circle`) based on its relative size comparing to the rest of the sub-image. Finally, the polar transformation of the sub-image of the original image is performed based on the center coordinate computed earlier. The transformation is parametrized by the angle granularity (`n_angles`), profile length (`plen`), and the length of ignored part of the profile (`ignored_len`).

Visualization in Figure 5.4 demonstrates intermediate and final results of the star washer segmentation routine, referenced with respect to the computational graph nodes.

5.8.2 Sharpness measurement

Another example of a computational graph is the algorithm for sharpness measurement on the custom calibration object, used in conjunction with a robot control program in paper 5.

The original image is supplied in the grayscale format. First, it undergoes thresholding operation to highlight the light regions on the image (including the white background of the calibration object). The thresholded binary image is eroded to remove the influence of the black lines in well-focused images. Further, connected components are identified. The goal is to segment the connected component belonging to the calibration object. To do that, a filter based on width-to-height ratio range and minimal region area is applied. The selected region of interest is cropped from the original image, and is used as an input to the Sobel operator in the x direction. From the middle of resulting gradient image (in terms of y axis), a horizontal line profile is extracted. As a measure of sharpness, the standard deviation of this profile is used: the more the original image is in focus, the greater variability between dark and bright pixels in the calibration region.

Figure 5.6 demonstrates the intermediate results (`sobelx_im` and `profile_sx`) of the sharpness measurement routine for images with different positioning of the calibration object.

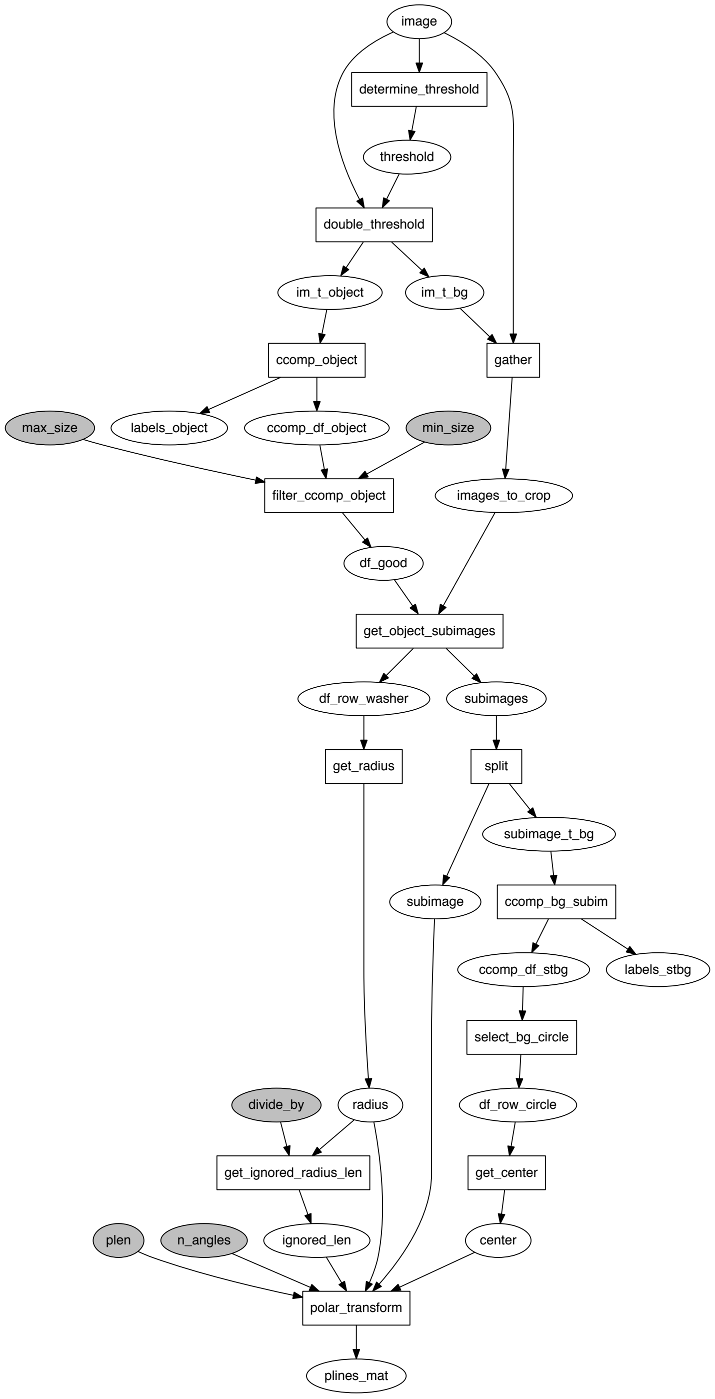


Figure 5.3: Star washer image analysis computational graph

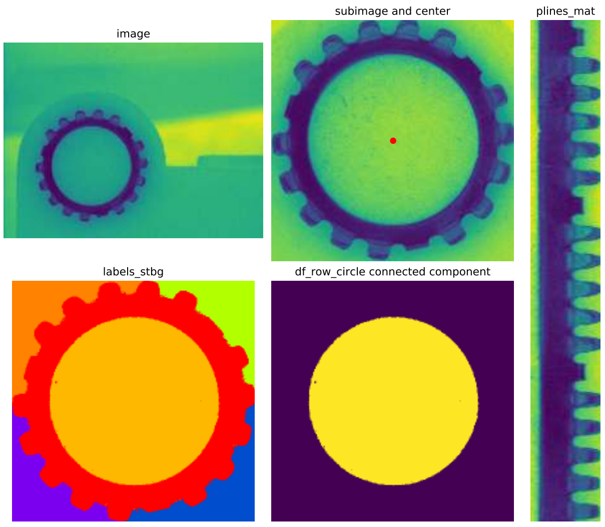


Figure 5.4: Visualization of tokens from the star washer computational graph

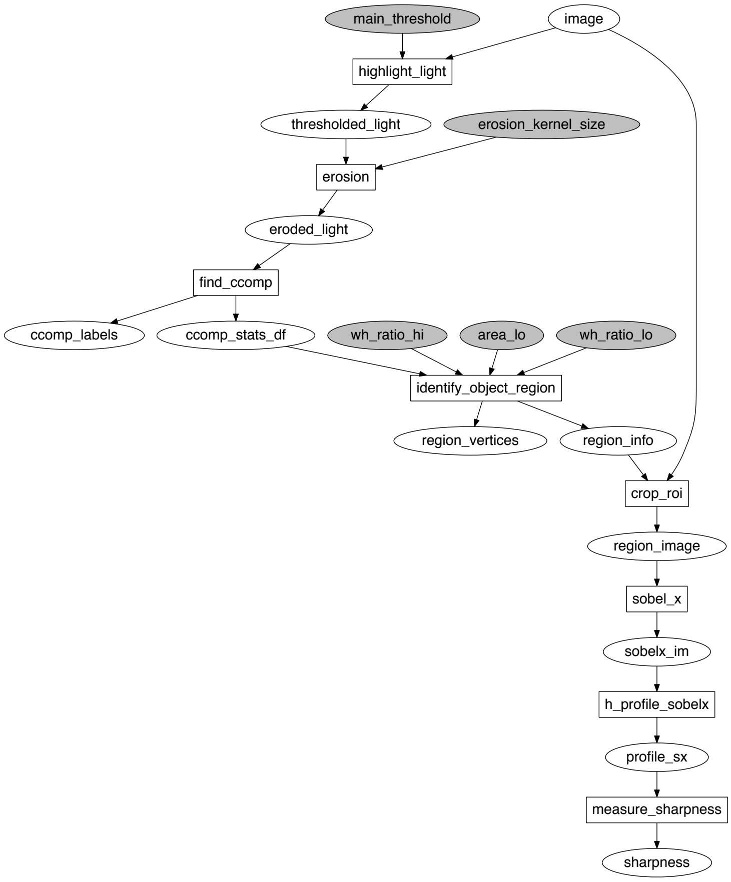


Figure 5.5: Computational graph for sharpness measurement

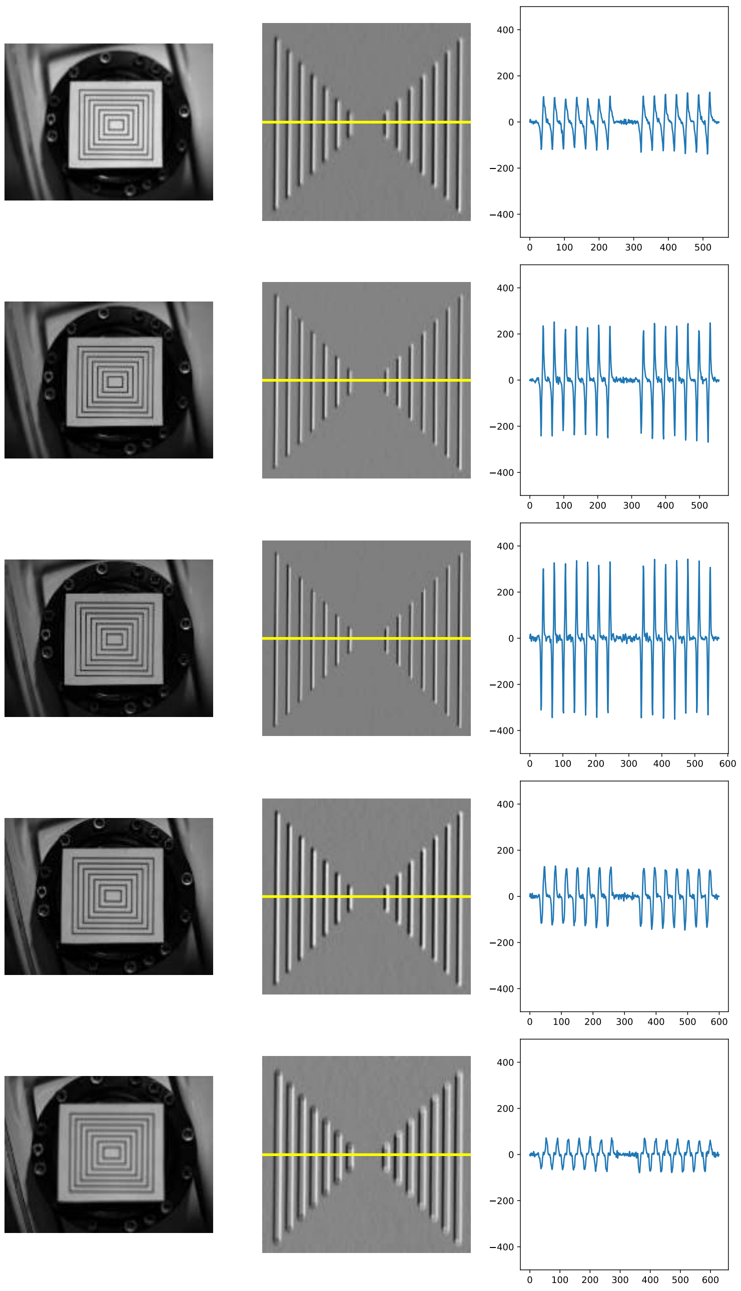


Figure 5.6: Visualization of sharpness measurement intermediate results for images with different positioning of the calibration object

Chapter 6

Image acquisition

Image capture and image acquisition are two integral processes that happen before an image becomes available for the processing algorithm. Image capture happens when light rays come through the lenses and get detected by the image sensor as a spatially distributed electrical signal. Image acquisition is concerned with transferring the digitized image into the computer memory, normally over some communication medium.

In industrial environments, cameras based on the GigE Vision acquisition standard are widely used. This standard is specifically suitable for engineering distributed vision systems, as it allows to connect computing equipment to multiple cameras using the available Ethernet equipment. The GigE Vision standard is widely supported by commercial vision software. In addition, camera vendors provide low-level software development kits that allow building custom vision solutions.

Using commercial vision software to realize the image acquisition function is in many cases too inflexible. At the same time, the approach of using SDKs requires building custom adapters to the application-level code. From the software engineering perspective, it is of interest to possess a service-oriented solution for the image acquisition function.

This chapters introduces the general principles of the image acquisition function, with the focus on the used in this PhD project GigE Vision standard. Furthermore, an overview of service-oriented architecture is presented, followed by a glimpse into the way the flexible image acquisition service (FxIS) has been developed.

6.1 Technical details of image acquisition

Industrial cameras can be subdivided into two classes: *asynchronous reset cameras*, which perform capture only when synchronized with an external event, and *free-running cameras*, which continuously capture images at a constant rate. The latter are more commonly used, and have two acquisition modes that can be realized. Depending on the design of the image acquisition module, the caller may issue a trigger event and process the acquired image sequentially (the synchronous mode), or perform triggering and processing concurrently (the asynchronous mode) [53]. The latter case is more effective, since it allows to process each individual image the camera is able to capture, and involves several threads of execution.

A camera is constrained by its maximal framerate r, measured in frames per second (fps). The same performance can be expressed with the inversely proportional inter-arrival time $\tau_{ia} = r^{-1}$. For example, Prosilica GC1350 cameras, used in paper 4, offer $r = 20$ fps, which result in $\tau_{ia} = 50$ ms.

Once the image is acquired, it is laid-down contiguously (row-by-row) in the computer memory as an array of `unsigned char`. An image sensor having the dimension of $w \times h$ and c color channels (for monochrome and Bayer images, $c = 1$), produces the array of size hwc.

6.2 GigE Vision

There exist several camera communication standards used as a backbone of the image acquisition function:

1. Camera Link
2. CoaXPress
3. IEEE1394
4. USB 3 Vision
5. GigE Vision

GigE Vision is based on the widely-used Gigabit Ethernet physical/link layer networking protocol. It provides a number of advantages over other standard, such as the ability to design distributed network topologies (as opposed to point-to-point), scalability in terms of adding more cameras into the systems, ability to work on standard hardware and software, and bandwidth suitable for real time video streaming [55].

The GigE Vision standard is based on UDP, hence allowing for higher speed of image transmission as compared to TCP. On top of it, two application-specific protocols are defined, GigE Vision Control Protocol (GVCP) and GigE Vision Streaming Protocol (GVSP). These protocols increase transmission reliability as compared to the raw UDP. Additional research has been done in improvement of packet loss during UDP-based communication in GigE Vision systems [56].

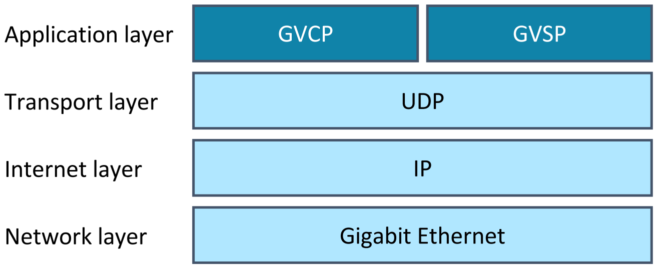


Figure 6.1: GigE Vision protocol stack

This PhD project utilized a number of *Allied Vision* GigE Vision cameras (see Section 2.4). These cameras are supplied with the vendor's SDK, dubbed *Vimba*, that allows to develop applications with image acquisition capabilities in C, C++, and for the .NET platform.

6.3 Service-oriented architecture

Service-oriented architecture (SOA) is a software architecture design pattern based on discrete, normally distributed, software modules that provide application functionality as service to other applications.

Service-oriented computing is important theme of research and development in information systems and software engineering when it comes to realization of distributed systems. Examples of protocols and paradigms realizing SOA are the following:

- SOAP: XML-based protocol for strictly typed machine-to-machine communication over HTTP.
- REST: an architectural style for web-services based directly on HTTP protocol, allowing to create, retrieve, update and delete resources.
- Publish/subscribe architecture: asynchronous messaging approach based on a message queue used by subscribers and publishers of messages.

- Microservices: architectural style of partitioning a single application into a collection of loosely-coupled services.

The above approaches have somewhat different technical, semantic and behavioral characteristics. Nevertheless, they share the common principle of decomposition of computational units and provision of services on demand. In the field of industrial automation, SOA manifested itself in such forms as OPC-UA, DPWS, the Tweeting Factory [57], and services and topics in the Robot Operating System (ROS).

6.4 FxIS

This section describes the principles and the main components the Flexible Image Acquisition Service (FxIS) presented in paper 4.

The principle structure of an FxIS application is shown on Figure 6.2. The highest-level component is a `Service`. It is responsible for handling requests to the imaging system from various clients and holding the control over the image acquisition process. Per each camera connection, there exists a `Streaming` object, running in a separate thread, and an `ImageStream` concurrent data structure.

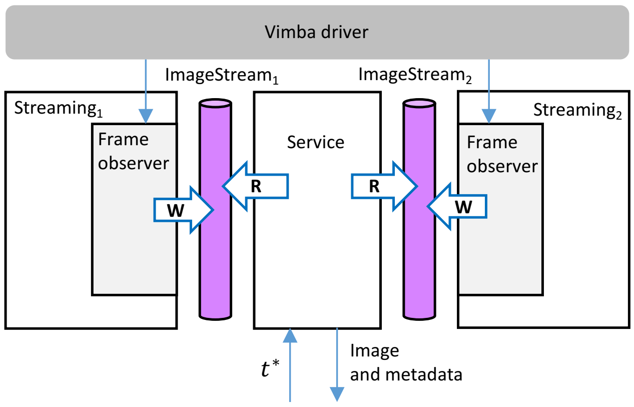


Figure 6.2: FxIS application components

`ImageStream` constitutes a form of a circular buffer with frames annotated by their arrival timestamps, providing the functionality of image storage and search-based retrieval. This data structure can be safely accessed by multiple threads. While the camera driver writes (W) to the camera's `ImageStream`, the `Service` component may request (R) images from the stream with the current timestamp t^*.

The programming model in Vimba C++ driver is based on the Observer design pattern. A concrete frame observer class, enclosed in an instance of `Streaming`, is associated with the given camera, and the driver invokes the `FrameReceived` method once a new frame is acquired. In the free-running mode it happens every τ_{ia}.

The dynamic workflow of an FxIS application is shown on Figure 6.3. When started, a `Service` spawns a number of threads (each per one camera interface) based on the corresponding `Streaming` objects. When instantiated, the latter establish connections with the respective cameras, initiate image capture, and start acquisition of each frame. When the work time of an FxIS service is finished (e.g. because of the user's request), all the threads are signaled for completion via instances of the `BlockingWait` and `EventObject` concurrent data structures $\{bw_i, eo_i\}$.

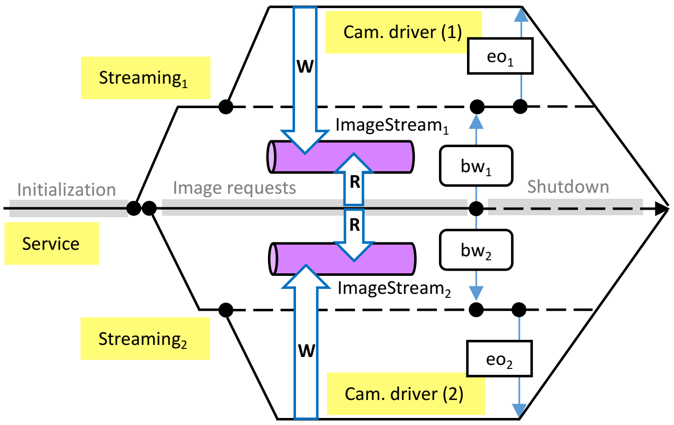


Figure 6.3: Threads and concurrent data structures' interaction in an FxIS application

An instance of `BlockingWait`, let's denote it as `bw`, is shared by two threads. Thread 1 calls `bw.wait()`, which lead to blocking until thread 2 calls `bw.notify()`. An instance `eo` of `EventObject` is also used for synchronization between two threads, but with a different interaction pattern. The producer thread calls `eo.hasOccured()` when a particular event has occurred (in the FxIS work flow example, the service stop requested). The consumer of `eo`, in turn, repeatedly checks for event, e.g. in context of infinite loop:

```
while (true) {
  // ... Primary logic of the loop
  if (ready.hasOccured()) {
    break;
  }
}
```

The concurrent access in `ImageStream` is secured by the internal mutex-based implementation. When `ImageStream::getImage` is invoked, the thread blocks and wait until the next frame arrives from the camera driver. This ensures that, if $(t^{(\text{next image})} - t^*) < \tau_{ia}/2$, the next image will not be missed. After the thread is unblocked, the nearest timestamp search is performed, and a copy of the corresponding `cv::Mat` image is returned to the caller.

Chapter 7

Control and programming of industrial robots

An industrial robot constitutes a reprogrammable multi-functional manipulator, able to perform a variety of tasks through variable programmed motion. Robots are widely applied in automated manufacturing, and are supplied as end-products with carefully engineered hardware and software components. A robot controller operate under a vendor-specific real-time operating system, which provide the proprietary programming language to realize the operational logic. It is a standard practice to program all the logic to be run by the robot controller. This is an efficient approach, although it makes the resulting robot logic more rigid and difficult to integrate into a system with distributed components.

Paper 5 proposes an alternative approach, where the robot controller provides a TCP/IP server with a collection of robot skills, and the dedicated control module communicates commands to the robot server and participates in a publish-subscribe communication with other distributed components. The control module is built using the `pyadept` library, which utilizes Python AsyncIO coroutines. The computational abstractions in `pyadept` allow for composing communication-heavy control logic in an event-driven way.

This chapter starts by presenting the robot geometry formalisms and basic motion commands available in most robot controllers. Since the practical work in this thesis is done with an Adept Viper s850 robot, the underlying Adept V+ platform is briefly presented. The remainder of the chapter covers the principles of event-driven thinking, cooperative multitasking with coroutines, and robotic middleware.

7.1 Industrial robot geometry

A robot manipulator is composed of links connected by joints. All possible configurations of the joints q_i comprise the configuration space $\mathcal{C}$ of the robot (q_i can correspond to joint angle for a revolute joint or joint offset for a prismatic joint). Thus, for a standard 6-axis manipulator, the configuration vector is defined as $(q_1, q_2, ..., q_6)^T \in \mathcal{C}$, where $\mathcal{C} \subset \mathbb{R}^6$.

A central notion in both robotics and computer vision is a pose, an object describing position and orientation of one Cartesian coordinate frame in relation to the other. Thus, a pose ${}^A\xi_B$, describing position and orientation of coordinate frame $\{B\}$ in terms of coordinate frame $\{A\}$ is defined as a tuple:

$$
{}^A\xi_B = ({}^A R_B, {}^A t_B) \tag{7.1}
$$

where ${}^A R_B$ and ${}^A t_B$ represent rotation and translation respectively.

In $\mathbb{R}^3$ translation can be defined by a vector $\mathbf{t} \in \mathbb{R}^3$, while rotation has several alternative forms of representation: a 3×3 special orthogonal matrix ($\mathbf{R} \in SO(3)$), a quaternion, or a three-angle representation [58]. It is common to combine a rotation matrix $\mathbf{R}$ a translation vector $\mathbf{t}$ into a single homogeneous transformation matrix $\mathbf{T} \in SE(3)$:

$$
\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0_{1\times3}} & 1 \end{bmatrix} = \begin{bmatrix} | & | & | & t_x \\ \mathbf{r_x} & \mathbf{r_y} & \mathbf{r_z} & t_y \\ | & | & | & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.2}
$$

Combined, a pose has 6 degrees of freedom: 3 for rotation and 3 for translation. In case of using $SE(3)$, they constitute $(\mathbf{r_x}, \mathbf{r_y}, \mathbf{r_z}, t_x, t_y, t_z)$.

In robotic applications one considers a chain of poses with the first one being in terms of the base coordinate frame $\{B\}$, and the last one being the pose of the end effector coordinate frame $\{E\}$ in terms of the coordinate frame of the last joint $\{6\}$:

$$
{}^B\xi_1 \rightarrow {}^1\xi_2 \rightarrow {}^2\xi_3 \rightarrow {}^3\xi_4 \rightarrow {}^4\xi_5 \rightarrow {}^5\xi_6 \rightarrow {}^6\xi_E \tag{7.3}
$$

Because the essence of a robot's operation is motion of its end-effector, achieved by combined motion of joints, one defines a *task space* $\mathcal{T}$ as a set of all possible poses of the end effector with respect to the robot's base coordinate frame, i.e. ${}^B\xi_E \in \mathcal{T}$.

Figure 7.1 shows the two coordinate frames of interest on an Adept Viper s850 industrial robot.

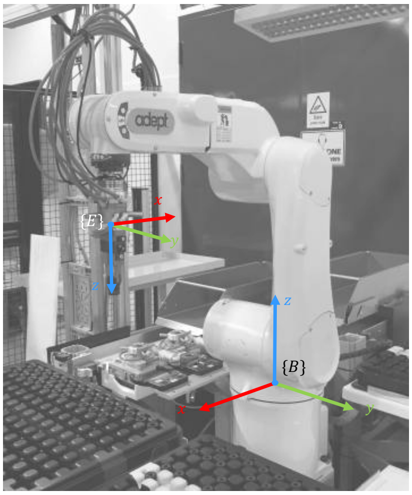


Figure 7.1: Base and end effector coordinate frames in Adept Viper s850

7.2 Robot commands

In sensor-based robotics, especially with custom-built robots in the research environment, it is common to directly control individual robot joints. In this setting, the robot's configuration space $\mathcal{C}$ is manipulated in a closed loop by using feedback from computational nodes processing the sensor data (e.g. using visual servoing).

With industrial robots it is more common to issue commands with respect to the task space $\mathcal{T}$, which are internally handled by the robot controller, i.e. the latter solves the inverse kinematics problem.

Point-to-point motion from ${}^{B}\xi_{P_1}$ to ${}^{B}\xi_{P_2}$ is started when the robot is in ${}^{B}\xi_{P_1}$, and can be specified with either ${}^{B}\xi_{P_2}$ or ${}^{P_1}\xi_{P_2}$ (for relative movement). *Linear motion* from ${}^{B}\xi_{P_1}$ to ${}^{B}\xi_{P_2}$ is specified similarly to point-to-point, but the robot controller ensures straight line trajectory of the motion. *Motion relative to the end effector coordinate frame* is applied for fine-grained end-effector control. Another important command is *breaking*. If two subsequent motion commands MC_i and MC_{i+1} are not separated by breaking, the combined movement will be interpolated.

In the work underlying paper 5, the described robot commands are defined as skills in the robot controller and invoked by sending messages over a TCP/IP connection. For more details regarding the communication protocol used for commands invocation, refer to paper 5.

7.3 Adept V+ platform

An Adept robot controller runs V+, a real-time multi-tasking operating system. It controls robot motion, input/output, task management, and other necessary operations. The Adept Viper s850 robot, used in the work underlying this thesis, is controlled by Adept SmartController CX running V+. I/O interfaces of this robot controller include digital signals, DeviceNet, Ethernet, and RS-232.

V+ is also the name of a programming language for the Adept platform. V+ codebase is comprised of subroutines called *programs* that are gathered in *modules*. An individual program can be associated with an operating system *task*, which in turn can be run concurrently with other tasks. To program an Adept robot controller, one requires a client programming environment (such as Adept ACE, Adept DeskTop), that runs on a Windows-based programming station.

Paper 5 describes a two component architecture for flexible control of an Adept robot, comprised of a server running on Adept SmartController CX and a Python-based master control node deployed to a Raspberry Pi.

7.4 Event-driven systems

One distinguishes between time-driven and event-driven systems. In the former case, everything is modeled with respect to a clock with given frequency. For example, a continuous signal is sampled at a constant time interval. Such systems are a subject of study in classical control and constitute a natural choice for modeling physical processes, when the state of the system changes continuously in time. Conversely, event-driven paradigm presumes that a system is characterized by a discrete state space, and events can occur at any time instant. An *event* in this case constitutes any instantaneous occurrence that causes transition from one system state to another [17].

The behavior of industrial automation systems is to a large extent event-driven. It manifests itself in both the nature of the applications and in the inherent properties of digital communication systems.

In control engineering practice that deals with time-driven systems, an event is defined as a change in a boolean signal or, given a more complicated time-sampled signal, an event is said to have occurred when a signal spike occurs, e.g. based on a defined threshold.

On the operating system level, an *I/O event* is associated with a particular resource becoming ready, e.g. data has arrived from the network and is ready for non-blocking access. In distributed systems, events can be handled as messages communicated in a publish-subscribe environment. For example, Hinze et al. [59] define an event as a message with a timestamp that can signalize either a change of a system state or a notification from an observation.

A lifecycle of an event in the context of computational systems consists in many cases of the following stages (adapted from [60]):

1. Event occurs at time instant t. In most cases t cannot be measured directly.

2. Event is captured at time instant $t_{capture}$. Since $t_{capture}$ is typically the first time instant that can be directly measured, the goal is that $(t_{capture} - t) \to 0$.

3. Event is queued at time instant t_q. This implies that a data object describing the event (or the corresponding serialized byte string) is put onto a queue or written to a socket.

4. Event is polled at time instant t_{poll}. Queuing and polling can happed either on one computational node with several threads or processes, or two endpoints that communicate over the network.

5. Event is processed at time instant $t_{process}$.

In paper 5, the event-driven design is applied in `pyadept` using Python AsyncIO coroutines. The latter are briefly presented in the subsequent section.

7.5 Coroutines and AsyncIO

A *coroutine* constitute a software abstraction that generalizes a subroutine by extending it with the capabilities of suspending and resuming at specified code locations.

A set of coroutines, each representing a particular application-specific task, are managed by en event loop to realize *cooperative multitasking*. This entails that each coroutine object performs a part of its computation, suspends and yields control to the event loop or other coroutine object. Such programming style gives the most evident advantage when dealing with I/O events.

Coroutines in Python are realized with the AsyncIO framework, which is a part of the standard library since Python 3.4. In addition to the core coroutine functionality, AsyncIO provides a set of non-blocking I/O abstractions. In `pyadept`, described in paper 5, TCP/IP communication is realized using `StreamReader` and `StreamWriter` objects.

7.6 Robotic middleware

Messaging middleware allows to decouple the communicating components by introducing message queuing, built-in address resolution (e.g. via handling logical addresses such as topic names), and usage of a common data serialization format [61]. An important feature of middleware is the provision of the publish/subscribe and other messaging patterns, which allows to design a distributed system in event-driven fashion.

The defining components of a particular middleware solution are the communication protocol (transport-level TCP and UDP, wire-level AMQP, ZeroMQ/ZMTP, MQTT), the communication styles (request/reply, publish/subscribe), and the data serialization method (typically realized with an interface definition language like Protobuf or Apache Thrift). Many middleware solutions are based on a central broker, e.g. ActiveMQ and RabbitMQ. ZeroMQ is an example of broker-less middleware, in which the message queuing logic runs locally within each communicating component [62].

Robot Operating System (ROS) is the most widely used middleware that is specifically designed for building distributed robotic systems. It supports request/reply remote procedure calls via *services*, and publish/subscribe communication via *topics*. Messages in ROS are serialized with the built-in serialization mechanism. A ROS system requires a central master server, responsible for name resolution. On the transport layer, both TCP and UDP are supported via standard sockets. In addition to the communication capabilities, ROS provides a pool of maintained community-developed algorithms.

In its current form, ROS is tightly coupled with Ubuntu as the runtime platform. ROS2 is an ongoing project aimed at adapting the ROS functionality to cross-platform environments (including for small embedded platforms), as well supporting real-time control capabilities [63]. ROS2 utilizes Data Distribution Service (DDS) middleware for communication.

Another example of robotic middleware is YARP [64]. For efficient inter-process communication with minimum latency, the `ach` Linux kernel module has been developed [65, 66].

When it comes to middleware, `pyadept` relies on ZeroMQ for realization of publish-subscribe communication. The goal of paper 5 is not to introduce a new robotic middleware, but rather to validate general principles in the a flexible way with lightweight tools.

Chapter 8

Discussion and further work

This chapter outlooks the conducted work described in this thesis, and relates it to the research questions formulated in chapter 1. Based on this discussion, the planned prospects for the further work are outlined.

8.1 Answering the research questions

The following three research questions have been previously formulated:

RQ1 How to better couple the automated systems' perception and action with the means of computational abstractions and architectural solutions?

RQ2 How can the image acquisition function be exposed in a service-oriented manner?

RQ3 How to achieve greater composability of communication-heavy robot logic by utilizing discrete event nature of distributed robotic systems?

Paper 1 is not directly related to any research question. However, it sets an industrial background to the work done as part of this PhD project. The work done while preparing the paper involved a great deal of interaction with employees at the KA plant in Raufoss, as well as studying the production systems at the mentioned facility. It also centered the focus of practical undertakings of the PhD project on the task of star washer inspection.

The task of star washer inspection was not *fully* addressed in this PhD project, mainly due to rather high requirements for equipment availability. Nevertheless, focusing on star washers resulted in a range of contributions. Most importantly, paper 2 presented the envisioned principles of the prospective robot-based inspection cell, introduced a custom algorithm for processing images of star washers imaged from the top-down perspective, and tackled the problem of orientation classification for robotic picking using machine learning methods. To gather image data for the top-down inspection, an industrial camera with a 35 mm focal length lens was used. This setup allows close-range imaging, but is very sensitive with respect to the pose of the object relative to the focus plane. The hassle of manual alighting the camera and the object motivated the need for automatic focus plane calibration of the robot arm. This task was applied as a use case in paper 5.

RQ1 has targeted development of systems with discrete event behavioral semantics and perceptual capabilities based on machine vision. The first steps in this direction were made in [5] (not included as a part of this thesis) by formulation of the Discrete Event Data Flow formalism, aimed at combining dynamic data flow graphs with discrete event semantics (as used in publish-subscribe systems). During the paper's presentation at the conference and discussions with fellow researchers in industrial automation and computer vision, it became evident that overly formalized approach of DEDF is too cumbersome, especially in the prototyping phase.

Nevertheless, the idea of *reactive pipelines* have further evolved in the implementation-oriented direction and led to development of the EPypes framework, presented in paper 3. Many of the motivations leading to design solutions in EPypes had arisen during practical activities in the laboratory, as well as while developing vision algorithms and analyzing data. EPypes was thus designed to accommodate both flexibility in system development and well-defined structure.

It was shown in paper 1 that the current status of automated manufacturing systems and the associated machine vision solutions in the case company is highly rigid due to high-speed/high-volume production requirements. However, the company's interests regarding continuous monitoring of the vision systems' operation and automatic adjustments of operating parameters constitutes the capability that should be tackled with a much improved industrial-grade version of EPypes.

RQ2 was motivated by the difficulties with using Ethernet-based industrial cameras in a service-oriented context. Similarly to EPypes, there was a need for a more flexible tool with well-understood internal behavior. Due to the availability of Allied Vision cameras, it was decided to apply the vendor's own low-level camera SDK, dubbed Vimba, for development of the envisioned flexible image acquisition service. The latter was presented in paper 4.

To complete implementation of all the parts of the planned distributed system for robot-vision interaction (see Figure 2.3), the components responsible for robot control had to be developed. This was a great platform for working on **RQ3** and investigating the possibilities for composable design of communication-heavy robot logic. The results in this direction were presented in paper 5.

8.2 Lessons learned and further work

One of the most tedious aspects of the practical laboratory work has been software deployment to the distributed computational nodes, as well as debugging in the distributed setting. The VLAN-based network configuration (paper 5), where different nodes were part of different subnets, made the deployment even more challenging. To focus on the important things and not on deployment tools development, the "vanilla" Unix tools like `rsync` and `scp` were mainly used. In the future, better deployment strategies have to be devised. They may be based on Linux container technologies and DevOps tools, used traditionally in cloud computing settings.

The difficulties with deployment led to better understanding of the issue of complexity in management distributed control systems. This complexity is what hinders novel control solutions from industrial adoption, and what needs to be addressed to facilitate the progress of Industry 4.0 and the related initiatives.

One of the takeaways from working on this PhD project is that focusing on implementing concrete applications is more fruitful than premature theorizing. It also became clear why Python had become so widespread in the robotics research community. A substantial benefit is the ability to quickly validate the ideas and rapidly iterate towards a workable solution. In addition to making the Python ecosystem the bedrock of most of the software projects (see section 2.4), an important decision has been to use lightweight communication machinery (ZeroMQ, Protobuf, and raw TCP) instead of sticking with ROS. This allowed to be flexible across different platforms (Ubuntu, Raspbian, macOS, Windows) and greatly improved the development pace. At the same time, it is wise to strive for integration of the developed projects into the pool of open source code that is widely used by the research community. In the robotics context, ROS is the most popular platform. As the architecture and functionality of the projects presented in this thesis had become more stable, the strategy of integration into ROS became clear. As described in paper 5, the currently in-progress ROS2 initiative is a suitable ecosystem to target.

In the future work, it can be of interest to switch the focus *away* from flexibility-orientation. It is natural for vision application to strive for efficiency. It is also well-known that flexibility and efficiency are often competing factors, which is the reason why the manufacturing industry traditionally adopts simpler and more reliable (but less flexible) systems. For the application context described in this thesis, it can be interesting to integrate the developed tools with technologies such as FPGAs, PLCs, and fieldbus networks.

In its current state, EPypes targets very specific class of vision algorithms, which are the ones operating on a single image or several related images. What is left out is the analysis of real-time video streams, which requires maintaining a chain of frame-to-frame measurements, tackle the issues with outliers, and perform probabilistic estimations. This area should definitely be tackled, with new abstractions based on both EPypes and FxIS.

In the presented work, database systems did not play a key role, although the long-term vision in paper 1 included considerations regarding databases. The future work should put this vision into action by practical validation of use cases where systems based on EPypes, PyAdept, and FxIS would bring additional value when combined with databases.

FxIS has a large room for improvements. In its current form, it supports requests for a single image per camera, and works only with Allied Vision cameras via the Vimba SDK. The plans for new features include integration with other camera platforms, such as Basler Pylon and `raspicam`, implementation of additional modes of image request (e.g. several images per camera), and optimization of internal operation.

The tools developed in this project can greatly accelerate various research tasks involving combination of robots and cameras. As noted in section 2.6, the initially envisioned scope of this thesis included geometric computer vision tasks. Even though the early research undertakings had to be put on hold, they can now continue at a faster pace. Other obvious research direction is further work on star washer inspection. For this to be fruitful, several hardware addition have to be made to the robot laboratory, including lighting and gripping equipment.

EPypes, as well as its conceptual predecessor DEDF [5], were conceived with specific application area in mind, which constituted vision-guided industrial robot setups with distributed computational nodes. The initial focus was largely on a well-defined structure. However, while developing EPypes, another important value proposition was uncovered, which is assistance in early stage algorithm development. This aspect was greatly highlighted in paper 3 and validated throughout development of algorithms that became a part of this thesis. In the future it is of interest to apply EPypes for machine learning problems and other data science tasks.

Bibliography

[1] Oleksandr Semeniuta, Sebastian Dransfeld, Kristian Martinsen, and Petter Falkman. Towards increased intelligence and automatic improvement in industrial vision systems. *Procedia CIRP*, 67:256–261, 2018. ISSN 22128271. doi: 10.1016/j.procir.2017.12.209.

[2] Oleksandr Semeniuta, Sebastian Dransfeld, and Petter Falkman. Vision-based robotic system for picking and inspection of small automotive components. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 549–554. IEEE, aug 2016. ISBN 978-1-5090-2409-4. doi: 10.1109/COASE.2016.7743452.

[3] Oleksandr Semeniuta and Petter Falkman. Flexible image acquisition service for distributed robotic systems. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*, pages 106–112. IEEE, jan 2018. ISBN 978-1-5386-4652-6. doi: 10.1109/IRC.2018.00024.

[4] Oleksandr Semeniuta. Analysis of camera calibration with respect to measurement accuracy. *Procedia CIRP*, 41:765–770, 2016. ISSN 2212-8271. doi: http://dx.doi.org/10.1016/j.procir.2015.12.108.

[5] Oleksandr Semeniuta and Petter Falkman. Discrete event dataflow as a formal approach to specification of industrial vision systems. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, volume 2015-Octob, pages 849–854. IEEE, aug 2015. ISBN 978-1-4673-8183-3. doi: 10.1109/CoASE.2015.7294187.

[6] Ivanna Baturynska, Oleksandr Semeniuta, and Kristian Martinsen. Optimization of process parameters for powder bed fusion additive manufacturing by combination of machine learning and finite element method: A conceptual framework. *Procedia CIRP*, 67:227–232, 2018. ISSN 22128271. doi: 10.1016/j.procir.2017.12.204.

[7] Ken Goldberg. What is automation?, 2012. URL http://www.ieee-ras.org/images/publications/t-ase/t-ase-editorial-january-2012.pdf. Accessed: 2016-12-01.

[8] Amnon H. Eden. Three paradigms of computer science. *Minds and Machines*, 17(2):135–167, 2007. ISSN 09246495. doi: 10.1007/s11023-007-9060-8.

[9] E A Lee and S A Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. MIT Press, 2017. ISBN 9780262533812.

[10] Edward Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015. ISSN 1424-8220. doi: 10.3390/s150304837.

[11] Peter Bull and Isaac Slavitt. Data science is software, 2016. URL https://www.youtube.com/watch?v=EKUy0TSLg04. Accessed: 2018-05-21.

[12] Henning Kagermann and Wolfgang Wahlster. Recommendations for implementing the strategic initiative industrie 4.0. final report of the industrie 4.0 working group. Technical Report April, acatech – National Academy of Science and Engineering, Frankfurt/Main, 2013.

[13] Peter C Evans and Marco Annunziata. *Industrial Internet: Pushing the Boundaries of Minds and Machines*. General Electric, 2012.

[14] Yan Lu, KC Morris, and Simon Frechette. Current standards landscape for smart manufacturing systems. Technical report, 2016. URL http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8107.pdf.

[15] The Linux Information Project. The unix philosophy: A brief introduction, 2006. URL http://www.linfo.org/unix_philosophy.html. Accessed: 2018-07-25.

[16] Oleksandr Semeniuta. Calibration of robot vision systems for flexible assembly, 2014.

[17] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008. ISBN 9780387333328.

[18] M.P. Groover. *Automation, Production Systems, and Computer-integrated Manufacturing*. Prentice Hall, 2008. ISBN 9780132393218.

[19] Serope Kalpakjian and Steven Schmid. *Manufacturing Engineering & Technology*. Pearson Prentice Hall, 7 edition, 2013.

[20] M. Santochi and G. Dini. Sensor technology in assembly systems. *CIRP Annals - Manufacturing Technology*, 47(2):503–524, 1998. ISSN 0007-8506. doi: http://dx.doi.org/10.1016/S0007-8506(07)63239-9.

[21] NF Edmondson and AH Redford. Generic flexible assembly system design. *Assembly automation*, 22(2):139–152, 2002.

[22] AH Redford. Materials handling for general purpose assembly. *The International Journal of Production Research*, 29(2):229–246, 1991.

[23] G. Rosati, M. Faccio, A. Carli, and A. Rossi. Convenience analysis and validation of a fully flexible assembly system. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–8, 2011. ISBN 1946-0740.

[24] Christian Finetto, Giulio Rosati, Maurizio Faccio, and Aldo Rossi. Implementation framework for a fully flexible assembly system (f-fas). *Assembly Automation*, 2015. ISSN 0144-5154. doi: 10.1108/AA-03-2014-022.

[25] S. Gottschlich, C. Ramos, and D. Lyons. Assembly and task planning: a taxonomy. *IEEE Robotics & Automation Magazine*, 1(3):4–12, September 1994. ISSN 1070-9932. doi: 10.1109/100.326723.

[26] Z M Bi and Lihui Wang. Advances in 3d data acquisition and processing for industrial applications. *Robotics and Computer-Integrated Manufacturing*, 26 (5):403–413, 2010. doi: http://dx.doi.org/10.1016/j.rcim.2010.03.003.

[27] S.J.D. Prince. *Computer Vision: Models Learning and Inference.* Cambridge University Press, 2012.

[28] Elias N. Malamas, Euripides G. M. Petrakis, Michalis Zervakis, Laurent Petit, and Jean-Didier Legat. A survey on industrial vision systems, applications and tools. *Image and Vision Computing*, 21(2):171–188, 2003. ISSN 0262-8856.

[29] H Golnabi and A Asadpour. Design and application of industrial machine vision systems. *Robotics and Computer-Integrated Manufacturing*, 23(6):630–637, 2007. doi: http://dx.doi.org/10.1016/j.rcim.2007.02.005.

[30] STEMMER IMAGING. *Imaging & Vision Handbook.* STEMMER IMAGING GmbH, Pucheim, Germany, 2013. ISBN 978-3-00-039657-5.

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

[32] Henrik I. Christensen and GregoryD. Hager. Sensing and estimation. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics SE - 5*, pages 87–107. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-23957-4. doi: 10.1007/978-3-540-30301-5_5.

[33] R. Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015. ISBN 9780262328463.

[34] Karl J. Åström and P. R. Kumar. Control: A perspective. *Automatica*, 50:3–43, 2014. ISSN 00051098. doi: 10.1016/j.automatica.2013.10.012.

[35] Matthias Riedl, Holger Zipper, Marco Meier, and Christian Diedrich. Cyber-physical systems alter automation architectures. *Annual Reviews in Control*, 38 (1):123–133, 2014. ISSN 1367-5788. doi: http://dx.doi.org/10.1016/j.arcontrol.2014.03.012.

[36] Lihui Wang, Martin Törngren, and Mauro Onori. Current status and advancement of cyber-physical systems in manufacturing. *Journal of Manufacturing Systems*, May 2015. ISSN 02786125. doi: 10.1016/j.jmsy.2015.04.008.

[37] Rob Pike. Concurrency is not parallelism, 2013. URL `http://blog.golang.org/concurrency-is-not-parallelism`.

[38] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78, 2012. ISSN 00010782. doi: 10.1145/2347736.2347755.

[39] Rob Schapire. What is machine learning? Technical report, Princeton University, 2008. URL `http://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf`.

[40] Tom M Mitchell. The discipline of machine learning. *Machine Learning*, 17 (July):1–7, 2006. ISSN 0264-0414. doi: 10.1080/026404199365326.

[41] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, 2013. ISBN 978-1-4614-7137-0. doi: 10.1007/978-1-4614-7138-7.

[42] Poul Erik Frandsen, Kristian Jonasson, Hans Bruun Nielsen, and Ole Tingleff. Unconstrained optimization. Technical report, Technical University of Denmark, 2004. URL `http://orbit.dtu.dk/files/5891060/imm3217.pdf`.

[43] K Madsen, H B Nielsen, and O Tingleff. Methods for non-linear least squares problems. Technical report, Technical University of Denmark, 2004. URL http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3215/pdf/imm3215.pdf.

[44] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*, volume 1 of *Springer Series in Statistics*. Springer New York, New York, NY, 2009. ISBN 978-0-387-84857-0. doi: 10.1007/b94608.

[45] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson-Engineering, 2007. ISBN 049508252X.

[46] Stanislava Soro and Wendi Heinzelman. A survey of visual sensor networks. *Advances in Multimedia*, 2009:21, 2009. doi: 10.1155/2009/640386.

[47] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979. ISSN 0018-9472. doi: 10.1109/TSMC.1979.4310076.

[48] Radford M. Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, (1977):355–368, 1998. ISSN 978-1-932432-41-1. doi: 10.1007/978-94-011-5014-9_12.

[49] Giorgos Sfikas, Christophoros Nikou, and Nikolaos Galatsanos. Robust image segmentation with mixtures of student's t-distributions. *Proceedings - International Conference on Image Processing, ICIP*, 1:273–276, 2006. ISSN 15224880. doi: 10.1109/ICIP.2007.4378944.

[50] Tony F. Chan, Jianhong Shen, and Luminita Vese. Variational pde models in image processing. *Notices AMS*, 50(1):14–26, 2003. URL http://www.ams.org/notices/200301/fea-shen-03.pdf.

[51] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[52] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[53] C. Steger, M. Ulrich, and C. Wiedemann. *Machine Vision Algorithms and Applications*. Wiley-VCH Textbook. Wiley-VCH, 2007. ISBN 9783527407347.

[54] Ethan Rublee and Gary Bradski. Orb - an efficient alternative to sift or surf. pages 2564–2571, 2011. ISSN 1550-5499. doi: 10.1109/ICCV.2011.6126544.

[55] John Phillips. Choosing the right video interface for military vision systems. In Nibir K. Dhar and Achyut K. Dutta, editors, *Proceedings Volume 9481, Image Sensing Technologies: Materials, Devices, Systems, and Applications II*, may 2015. doi: 10.1117/12.2176651.

[56] Khaled Taouil, Tarek Jellad, and Zied Chtourou. Enhanced packet loss recovery for real time pc-based gige vision avi systems. *International Journal of Communication Networks and Distributed Systems*, 14(4):433, 2015. ISSN 1754-3916. doi: 10.1504/IJCNDS.2015.069677.

[57] Alfred Theorin, Kristofer Bengtsson, Julien Provost, Michael Lieder, Charlotta Johnsson, Thomas Lundholm, and Bengt Lennartson. An event-driven manufacturing information system architecture for industry 4.0. *International Journal of Production Research*, 48(3):1–15, jul 2016. ISSN 0020-7543. doi: 10.1080/00207543.2016.1201604.

[58] Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer Science & Business Media, 2011.

[59] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS '09*, page 1. ACM Press, 2009. ISBN 9781605586656. doi: 10.1145/1619258.1619260.

[60] Stephen Marz and Brad Vander Zanden. Reducing power consumption and latency in mobile devices using an event stream model. *ACM Transactions on Embedded Computing Systems*, 16(1):1–24, oct 2016. ISSN 15399087. doi: 10.1145/2964203.

[61] L. Magnoni. Modern messaging for distributed sytems. *Journal of Physics: Conference Series*, 608(1):1–8, 2015. ISSN 17426596. doi: 10.1088/1742-6596/608/1/012038.

[62] ZeroMQ. Broker vs. brokerless, 2008. URL http://zeromq.org/whitepapers:brokerless. Accessed: 2018-02-28.

[63] Brian Gerkey. Why ros 2.0?, 2018. URL http://design.ros2.org/articles/why_ros2.html. Accessed: 2018-02-28.

[64] Lorenzo Natale, Ali Paikan, Marco Randazzo, and Daniele E. Domenichelli. The icub software architecture: Evolution and lessons learned. *Frontiers in Robotics and AI*, 3(April):1–21, 2016. ISSN 2296-9144. doi: 10.3389/frobt.2016.00024.

[65] Neil T Dantam, Daniel M Lofaro, Ayonga Hereid, Paul Y Oh, Aaron D Ames, and Mike Stilman. The ach library: A new framework for real-time communication. *Robotics & Automation Magazine, IEEE*, 22(1):76–85, 2015.

[66] Neil T. Dantam, Kim Bøndergaard, Mattias A. Johansson, Tobias Furuholm, and Lydia E. Kavraki. Unix philosophy and the real world: Control software for humanoid robots. *Frontiers in Robotics and AI*, 3(March):1–15, 2016. ISSN 2296-9144. doi: 10.3389/frobt.2016.00006.