From low level perception towards high level action planning

Dissertation

in order to obtain the doctoral degree
"Doctor rerum naturalium"
of the Georg-August-Universität Göttingen

in the Doctoral program
Ph.D. Programme in Computer Science (PCS) of
the Georg-August University School of Science (GAUSS)

submitted by
Simon Reich

of Iserlohn, Germany
(place of birth)

Georg-August-Universität Göttingen
Göttingen, Germany
August 2018

Thesis committee

Prof. Dr. Florentin Wörgötter,
Georg-August-Universität Göttingen, Faculty of Physics, Third Institute of Physics

Prof. Dr. Wolfgang May,
Georg-August-Universität Göttingen, Faculty of Mathematics and Computer Science, Institute of Computer Science

Members of the examination board

First Reviewer: **Prof. Dr. Florentin Wörgötter**,
Georg-August-Universität Göttingen, Faculty of Physics, Third Institute of Physics

Second Reviewer: **Prof. Dr. Wolfgang May**,
Georg-August-Universität Göttingen, Faculty of Mathematics and Computer Science, Institute of Computer Science

Other members of the examination board:

Prof. Dr. Jens Grabowski,
Georg-August-Universität Göttingen, Faculty of Mathematics and Computer Science, Institute of Computer Science

Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen, Faculty of Mathematics and Computer Science, Institute of Computer Science

Prof. Dr. Minija Tamošiūnaitė,
Vytautas Magnus University, Faculty of Informatics, Department of Systems' Analysis

Prof. Dr. Ramin Yahyapour,
Georg-August-Universität Göttingen, Faculty of Mathematics and Computer Science, Institute of Computer Science

Date of the oral examination: October 30th, 2018

The canonical version of this document is the electronic copy maintained in the Github repository by the author. At this time, it is maintained at:

https://github.com/simonreich/dissertation

Much of the code created as a result of the research in this thesis is freely available under a GPLv3 license:

https://git.physik3.gwdg.de

The code for the Edge-Preserving Filter (see Sec. 2.2.2) created as part of this thesis is freely available under a BSD license as part of the Open Source Computer Vision Library (OpenCV):

https://opencv.org

For other usage, contact thesis@simonreich.de.

From low level perception towards high level action planning

Abstract

Nowadays, robots become more and more integrated into everyday life. Smartphones, desktop computers, and even cars can be thought of as robots, even though probably not autonomous robots. Many discussions about the term "autonomy" have sparked in recent years and one expects from a robot the ability to learn correlations between its actions and the resulting changes in its environment. The robot acts inside the so called action-perception loop, where it acts, similar to a human being, on a scene and is also able to perceive the changes. In this work, two robot systems are built and analyzed in terms of their action-perception loop.

The first part focuses on the perception side. Here, we consider three robots: A flying one and two wheeled ones. These machines have omnidirectional cameras installed. The data acqiered from the sensor usually require preprocessing in real-time. For this purpose a filtering algorithm called Edge-Preserving Filter (EPF) is introduced. It achieves higher quality results than traditional local methods and compared to current global state-of-the art methods its runtime is about three magnitudes faster. EPF performs on any dimension and scales well with data size. This enables it to run on 2d images as well as 1d sensor data, e.g. an accelerometer or gyroscope. Afterwards, the processed data are utilized for pose tracking. Here, a novel Visual Odometry algorithm named Embedded Visual Odometry (EVO) is developed. All computations run in real-time on embedded hardware without external tracking or data link to an external computing station. It is shown that the setup performs appromximately twice as good as current state-of-the art systems. As the proposed framework is entirely bottom-up and runs on embedded hardware, it enables truly autonomous robots.

In the second part, the focus lies on the action side of the action-perception-loop. A general way of bootstrapping, learning, and execution of actions, which is called Semantic Event Chain (SEC) is analyzed. In this work, a novel extension, which allows for high level planning of robot actions, is introduced. First, pose

information, which is generated by a novel 3d geometric reasoning algorithm, is included into SECs. This bottom-up abstract layer enables defining preconditions for actions in a natural way, which in turn allows to compute a scene's affordance. Second, adding the postconditions of an action makes the robot estimate the outcome of an action. This leverages high level action planning using only low level methods. SECs are applied to both two-dimensional and three-dimensional image data. Due to their clear structure, SECs can be utilized to solve a wide range of different problems in everyday life.

In total, this work consists of the following novel contributions: An efficient denoising algorithm, a Visual Odometry algorithm for robot pose estimation, and a planning framework, which allows to solve complex action plans using bottom-up, low level data. Each of these contributions has been implemented in live systems and has been run in an online manner. For each algorithm quantitative evaluation on existing benchmarks to demonstrate state-of-the art perception and action is performed. This work enables robots to navigate in previously unknown and possibly unstructured environments and perform complex action planning.

Contents

1. Introduction **1**

1.1. Prelude . . . 1
1.2. Historic approach . . . 3
1.3. Motivation . . . 6

2. From low level towards high level perception in robots **9**

2.1. Introduction . . . 9
2.1.1. The state-of-the-art of denoising filters . . . 10
2.1.2. The state-of-the-art of Visual Odometry . . . 12
2.2. Methods . . . 18
2.2.1. Hardware setup . . . 19
2.2.2. Noise and outlier detection . . . 19
2.2.3. Visual Odometry algorithm . . . 24
2.3. Results . . . 33
2.3.1. Effect on denoising of different subwindow sizes . . . 34
2.3.2. Effect on denoising of different thresholds τ . . . 34
2.3.3. Denoising of 2d images . . . 36
2.3.4. Denoising of 1d sensor data . . . 39
2.3.5. Time performance of denoising algorithm . . . 44
2.3.6. Visual Odometry in simulation . . . 45
2.3.7. Externally tracked indoor flights . . . 50
2.3.8. Office indoor flight . . . 53
2.3.9. Time performance of Visual Odometry algorithm . . . 55
2.4. Discussion . . . 56
2.4.1. Edge-Preserving Filter . . . 56
2.4.2. Embedded Visual Odometry . . . 57

3. Action planning in robots **59**

3.1. Introduction . . . 59

3.2. Methods 62
3.2.1. Action categories 64
3.2.2. Semantic Event Chains 68
3.2.3. Enriched Semantic Event Chains 78
3.2.4. Structural information 83
3.2.5. Affordance of Semantic Event Chains 85
3.2.6. Using affordance for planning 92
3.3. Results 99
3.3.1. 3d geometric reasoning algorithm 99
3.3.2. Scene affordance 106
3.3.3. Using affordance for planning 110
3.4. Discussion 125
3.4.1. 3d geometric reasoning algorithm 125
3.4.2. Scene affordance 126
3.4.3. Using affordance for planning 127

4. Conclusion and outlook **131**

A. Appendices **149**
A.1. Edge-Preserving Filter in the continuous domain 149

List of Acronyms

AAV Autonomous Aerial Vehicle. 1, 15–17, 24, 45, 57

AI Artificial Intelligence. 3, 4, 6, 14, 61

ANN Artificial Neural Network. 11

CPU Central Processing Unit. 13, 15, 44

DMP Dynamic Movement Primitives. 96

DNN Deep Neural Network. 6, 15

DoF Degrees of Freedom. 26

DSLR Digital Single-Lens Reflex Camera. 59, 62

EKF Extended Kalman Filter. 24, 55, 133

EPF Edge-Preserving Filter. iii, ix, xv, xvi, 9, 11, 17, 20, 24, 40, 44, 56, 57

ESEC Enriched Semantic Event Chain. 78, 83, 85, 90, 91, 134

EVO Embedded Visual Odometry. iii, 17, 31, 57

GPS Global Positioning System. 9, 10, 15, 16, 53, 87, 131

GPU Graphics Processing Unit. 15, 44

IMU Inertial Measurement Unit. xi, xxiv, 13, 19, 33, 50, 52–54, 57, 132

PCA Principal Component Analysis. 87

PSNR Peak Signal-to-Noise-Ratio. xv, 20, 36, 39, 40

RFID Radio-Frequency Identification. 87

RMSE Root-Mean-Square Error. xv, 20, 36, 39, 40, 47, 52

ROS Robot Operating System. xxv, 19, 62, 96

SEC Semantic Event Chain. iii, iv, xii, xiii, xvi, xvii, 59, 62, 72, 74, 75, 78, 85, 87, 89, 90, 92, 94, 96, 97, 99, 106, 107, 113, 124, 126–128, 133

SEM Semantic Event Matrix. xii, 73–75, 94, 96, 99

SVM Support Vector Machine. 87

VO Visual Odometry. xi, xxiv–xxvi, 19, 33, 52–54, 57, 58, 132, 133

WLAN Wireless Local Area Network. 13, 16

List of Figures

1. Introduction

1.1. Die size of one transistor during the years 1970 – 2017 [80, 96]. . . 2

1.2. Moravec's Paradox in popular literature [82]. 5

1.3. Schematic diagram of the Action-Perception loop: A scene is recorded by sensors, second, the agent's cognition analyzes the input and forms a plan, which it executes via its actuators. These in turn act on the scene, where changes are again perceived by the sensors. The left side is therefore called "action side", and the right side is named "perception side". 7

2. From low level towards high level perception in robots

2.1. Even today, denoising remains a challenging task. The here proposed real-time denoising filter is called EPF. 11

2.2. The pictures show the robots developed in this work. On the left, there is the WheelPi robot: a three-wheeled ground-based robot. In Fig. 2.2b the FlyPi robot is shown. It is a flying robot utilizing a quadrotor design. Both robots are part of the MovingPi library. 13

2.3. Flowchart of the methods in this chapter and how they relate. Details are explained in Sec. 2.2. 18

2.4. Overview of the system structure. A detailed explanation of all steps is shown in Sec. 2.2.2. 20

2.5. Periodic mirrored boundary conditions are used for image subwindows. A red rectangle denotes borders of original image. . . . 21

2.6. On the left a grayscale image, which needs to be filtered, is shown. For visualization purposes 100 px (marked in red) are chosen for detailed analysis and plotted in the large graph. Each pixel has Gaussian noise (variance of 1) added, additionally pixel 10 contains an outlier. At pixel 50 there is a color edge. In blue the same pixels are shown after being processed by the filter. The left subplot contains one subwindow sized 9×1 px. Pixel 10 is smoothed out, since the mean pixelwise color distance δ_m is low and thus pixel 10 is identified as outlier. The right subgraph shows another subwindow, which detects a color edge. δ_m is greater than threshold τ and therefore no values are smoothed inside this subwindow. 25
2.7. Pipeline of proposed algorithm. Details are outlined in Sec. 2.2.3. Enlargements of images in b) and c) can be found in Fig. 2.8. . . . 26
2.8. Enlargement of example frames b) and c) from Fig. 2.7. 27
2.9. Sketch of a camera observing an object $\vec{o}$, which appears at position $\vec{o}'$ in the image plane (b). In Figure (c) the camera is pointed at a hyperbolic mirror. 29
2.10. Shown is the effect of different subwindow sizes on one data set: A one dimensional grayscale image containing a color edge at pixel 25 and one outlier at pixel 10. Detailed explanations are shown in text, see Sec. 2.3.1. 35
2.11. Shown is the effect of three different thresholds on one data set: A one dimensional grayscale image containing a color edge at pixel 25 and one outlier at pixel 10. Detailed explanations are shown in text, see Sec. 2.3.2. 37
2.12. Visual comparison of filter results. Quantitative results are shown in Tab. 2.1. Images taken from Berkeley Image Data Set [10]. . . . 38
2.13. Examples of different denoising algorithms on stepwise data. . . 41
2.14. Examples of different denoising algorithms on sawtooth data. . . 42
2.15. Examples of different denoising algorithms on sinusoidal data. . 43
2.16. Urban canyon and indoor scenario with sparse optical flow (visualized as green dots and lines). 45

2.17. Overview of the simulation results as computed by the EVO algorithm proposed here. It is compared to state-of-the-art SVO algorithm [44]. 46
2.18. Translation error x, y, and z of the "Urban Canyon" trajectory shown in Fig. 2.17a. 48
2.19. Translation error x, y, and z of the "Indoor" trajectory shown in Fig. 2.17b. 49
2.20. Qualitative examples of recorded target trajectory. 51
2.21. The robot started and landed at position $(0, 0)^T$ and flew a figure-of-eight around a central obstacle shown in gray. The trajectory (in green) shows the internal believe state of the robot (fusion of Visual Odometry (VO) and Inertial Measurement Unit (IMU)); it is 19.4 ± 0.1 m long. It took the robot 38.4 s to fly the track. The starting point is marked with a blue cross, the estimated landing position with a small red circle, while the real landing position was again at $(0, 0)^T$. 54

3. Action planning in robots

3.1. One of the two Kuka Lightweight Robots [18]. Connected to the robot arm is a three-fingered gripper. 60
3.2. Flowchart of the methods in this chapter and how they relate. Details are explained in Sec. 3.2. 63
3.3. Schematic example actions in the ontology are shown for the three categories. From each category only one action is shown. The objects are marked using the following convention: h = hand, m = main m.s = main support, p = primary, p.s = primary support, s = secondary, s.s = secondary support, l = load, and cont = container (taken from Reich, Aein, and Wörgötter [97]). 71
3.4. A visualization of an object graph. Computer vision identifies and separates objects and their relative structure to each other (left image). One Semantic Event Graph (right image) results directly from the structure. Please note that multiple roots for one graph are allowed. 73

3.5. An example showing a pushing action in the SEC domain. The first row shows a pictogram view of the action. The *main* object, denoted with "m", sits on top the "main support" and the robot is not touching the *main* object. In the second keyframe the robot touches the *main* object and pushes it to the right. The robot's trajectory is marked with a dashed line. However, this trajectory information is not encoded in the SEC. In the third keyframe the robot hand is removed from the *main* object. The middle row holds a graph representation of the touching and not-touching relations; touching relations are marked with a line. In the bottom row the graph is represented as Semantic Event Matrices (SEMs). All three matrices hold a lot of static information. Therefore, a short form, which removes all static information, is introduced. For this example one could also write: "main object – robot hand: N T N". 75
3.6. Frames from a robot demonstration: The robot picks an apple from a plate and places it on the table. The corresponding graph representation is given on the right side. 77
3.7. This is the same scene as shown in Fig. 3.5 — a robot pushing the *main* object along its support. Please note the coordinate system, which is used when the keyframe matrix on the right is enriched by relative pose information, see Eqn. (3.2). For clarity only two dimensions are used here (where $y = 0$). 79
3.8. Step-by-step explanation of the geometric reasoning algorithm. . 82
3.9. Only these three subgraphs may exist around the *main* object. Any graph structure, which contains at least a *main* object and its support, can be reduced to a series of these subgraphs. Any subgraph consists of the *main* object, its support, and up to one more object. 84
3.10. A scene, as recorded by a robot is analyzed and a graph structure is generated. As *main* object the plate is chosen by either human or higher level algorithms. For each object around the *main* object a subgraph is generated. 86

3.11. Action perception loop of the presented system. First, d) the scene is recorded by a) a computer vision system: Here object segmentation, recognition, tracking, and eventually SEC extraction takes place. The Semantic Event Chain, as well as a labeled Point Cloud, is given to the b) SEC planner. The planner creates a plan based on a goal provided by e) a human being. The plan is given to c) a robot, which in turn will try to execute it. When encountering an error, e.g. the touching relations have changed in an unexpected way, an error signal is returned. The plan is recomputed or, if no plan is found, the error signal is escalated to the human. 97

3.12. In a) the scene is recorded and the current semantic relations are extracted. b) The goal state to the planner consists of the preconditions of the goal action that is to be performed. a) and b) are given to the c) simulator: Here, it is checked whether the preconditions of the goal state are met. If so, the plan may be executed on the robot. If no branch is left to check and no plan is found, an error message is sent. Else, the tree is expanded in d). Each branch is simulated using the postconditions from Tab. 3.4. Then, all possible actions are appended to the branch as leaves. Lastly, in e) branches that contain loops or are too long are terminated. 98

3.13. Three different scenes are used to test the algorithm. They resemble cluttered kitchen scenarios as one might expect them in the real world. 100

3.14. Qualitative results for the geometrical reasoning method, scene 1. Recorded depth points on the objects are marked using white dots. The algorithm is applied to the object pair apple and red pedestal, and blue cup and box. For graphical purposes only the largest cluster is shown with a red arrow. Here, the arrow points from the apple downwards to the pedestal, which is the "forbidden" direction, if you want to lift the apple. 101

3.15. Qualitative results for the geometrical reasoning method for scene 2. For graphical purposes only the largest cluster is shown with a red arrow. 102

3.16. Qualitative results for the geometrical reasoning method for a cluttered scene. For graphical purposes only the largest cluster is shown with a red arrow. Please note the two red arrows in (c). Here, the two largest clusters are depicted. 103
3.17. Scenario 1: The red apple is being pushed to the pedestal, which is touched by another apple. 112
3.18. Scenario 2: The robot needs to put a cutting board on top of another plate. For this it needs to empty the board first. 116
3.19. Scenario 3: The robot needs to *pour liquid into a bowl*. Currently, the bowl is used for fruits and needs to be cleaned first. 119
3.20. Graph relation of the first keyframe as shown in Fig. 3.21a. The *main* object "Cucumber" is marked in red. 120
3.21. Scenario 4: The robot needs to cut a cucumber. At the beginning the cutting board is occupied by an apple, which must be removed first. 123
3.22. The structure resembles Fig. 3.11, but includes e) a symbolic planner. This planner includes high level knowledge as object properties or functions of objects. 128

List of Tables

2. From low level towards high level perception in robots

2.1. Root-Mean-Square Error (RMSE) and Peak Signal-to-Noise-Ratio (PSNR) computed on the Berkeley Data Set (500 images) and the Coco Data Set (40775 images). The first line "Original" refers to the not denoised image. The error is ± 0.01 for all values. 39

2.2. Comparison of RMSE and PSNR computed on three different scenarios: 1) an alternating line, 2) a sawtooth wave, and 3) a sinusoidal wave. To each scene three different noise types (Gaussian, salt-and-pepper (s&p), or both) are added, resulting in 9 different experiments. Each experiment is repeated 1000 times and averaged; the error is ± 0.1 for all values. 40

2.3. Time performance for images of different sizes. The test images were taken from the validation set of the Berkeley Segmentation Data Set and Benchmark [10]. 100 measurements were taken and averaged. The proposed EPF filter is compared to state-of-the-art algorithm BM3D [27] as shown in [111]. BM3D is, according to [111], one of the fastest recent methods. The error is ± 0.1 for all values. 44

2.4. Results of the two simulation scenes "Urban Canyon" and "Indoor" [137]. Shown is RMSE, which measures the total difference of the entire flight trajectory compared to the ground truth information. Displacement holds the euclidean distance between ground truth finish position and estimated finish position. 47

2.5. For each of the six trajectories (which are shown in Fig. 2.20) ten trials were performed and the averaged RMSE in the x-y-plane for these trials is shown. In "manual mode" the quadrocopter was moved manually on the trajectories to eliminate problems from flight control algorithms. In "flight mode" trials were performed in full flight mode. 52

2.6. Average time consumption in milliseconds by individual components of the algorithm on the data set. Comparison between run times on a laptop (Intel Core i7 (2.80 GHz) processor and the Raspberry Pi (ARM Cortex-A53). It is compared to the SVO algorithms results as shown in [44]. 55

2.7. PSNR values computed on the Berkeley data set for state-of-the-art methods (as shown in [52]) compared to the proposed EPF filter. 57

3. Action planning in robots

3.1. List of atomic actions as taken from [133]. More actions are listed as "Some (sic) dynamic versions of 17 – 26"; for example, the action "throw-in". According to [133] there are three different manipulation types (listed in the "Type" column): 1: Hand-only-actions; 2: Separation actions; 3: Release determined actions. Abbreviations in the "Goal" column are defined as follows: d: destroying; r: rearranging; c: constructing; t: taking-down; h: hiding; and b: breaking. 66

3.2. Summary of ontology of actions. Actions are divided into three categories and further into sub-categories. There can be more than one action in each sub-category. Taken from Reich, Aein, and Wörgötter [97]. 69

3.3. List of preconditions for atomic actions on the SEC level (action list as shown in [133]). A "✓" denotes that the structure is allowed, if the action needs to be executed; the actions marked with "-" are not allowed; "n" is used, where the structure is not applicable as the state of the *secondary* is of no relevance. The left three columns show preconditions for the *main* object. The right columns show the preconditions of the *secondary* object of an action. Please note that the action's *secondary* object turns into the *main* object of the subgraph. 89

3.4. List of postconditions for atomic actions on the SEC level (action list as shown in [133]). A "✓" denotes that the structure is a possible outcome, if the action needs to be executed; the actions marked with "-" are not allowed. 96
3.5. Results for scene 1, see Fig. 3.13a. 104
3.6. Results for scene 2, see Fig. 3.13b. 104
3.7. Results for scene 3, see Fig. 3.13c. 105
3.8. The different scenes are enlarged in Fig. 3.13. Please note that one cannot check the preconditions for some actions, e.g. stirring, knead which are related to the material of objects. These actions are denoted with "n"; they require high level object knowledge. A "✓" denotes executability of the action; the actions "-" were correctly computed as not possible to execute. 109

Acknowledgments

First of all, I would like to thank my supervisors Prof. Dr. Florentin Wörgötter and Prof. Dr. Wolfgang May for guiding me through my work by sharing their experiences with me and for countless hours of fruitful discussions without which this work would not have been successful. This work has been done in collaboration with Dr. Eren Erdal Aksoy, Dr. Jan-Matthias Braun, Dr. Alejandro Agostini, and Prof. Dr. Babette Dellen, so I am very thankful for their efforts, too.

Second, I would like to thank all my colleagues and friends for their direct and/or indirect input to my work and for having great time together. Many thanks go to Aisha Aamir, Dr. Alexey Abramov, Dr. Mohamad Javad Aein, Johannes Auth, Moritz Becker, Dr. Sakyasingha Dasgupta, Dr. Michael Fauth, Dennis Goldschmidt, Juliane Herpich, Sebastian Herzog, Dr. Tatyana Ivanovska, Dr. David Kappel, Prof. Dr. Tomas Kulvicius, Jannik Luboeinski, Timo Lüddecke, Prof. Dr. Poramate Manoonpong, Dr. Daniel Miner, Dr. Timo Nachstedt, Dr. Jeremie Papon, Dr. Mayte Bonilla Quintana, Dr. Jan Markus Schoeler, Mina Lilly Shibata, Prof. Dr. Minija Tamosiunaite, Florian Teich, Dr. Christian Tetzlaff, Dr. Xiaofeng Xiong, Erenus Yildiz, and Fatemeh Ziaeetabar. An especially big thanks to Ursula Hahn-Wörgötter who was always a big help. Next, I would like to thank all members of the Feinmechanikwerkstatt and Elektronikwerkstatt of the Third Insitute of Physics. Without their detailed knowledge, patience, and outstanding craftsmanship many projects would not have been possible. A big thanks to all other members of the Third Insitute of Physics for your support and help.

Third, I would like to thank all my Bachelor and Master students: Damian Bast, Lars Berscheid, Caroline Campbell, Philipp Dönges, Martin Heinemann, Georg Jahn, Johann Kalies, Daniel Kalin, Erik Schultheis, Maurice Seer, and Kevin Vorwerk. It was always a big pleasure working with you.

Acknowledgments

A special thanks goes to my parents without whom I would not have achieved all that in my life what I have now. Thank you all for the patience, understanding, support and being always by my side no matter what.

Thank you very much indeed!

Simon Reich
Göttingen, 2018.

List of related publications

S. Reich, M. Seer, L. Berscheid, F. Wörgötter, and J. Braun. "Omnidirectional visual odometry for flying robots using low-power hardware". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp*. Vol. 5. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 499–507

S. Reich, M. J. Aein, and F. Wörgötter. "Context Dependent Action Affordances and their Execution using an Ontology of Actions and 3D Geometric Reasoning". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp*. Vol. 5. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 218–229

S. Reich, F. Wörgötter, and B. Dellen. "A Real-Time Edge-Preserving Denoising Filter". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp*. Vol. 4. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 85–94

T. Ivanovska, **S. Reich**, R. Bevec, Z. Gosar, M. Tamosiunaite, A. Ude, and F. Wörgötter. "Visual Inspection And Error Detection In a Reconfigurable Robot Workcell: An Automotive Light Assembly Example". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp (VCEA)*. vol. 5. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 607–615

S. Reich, A. Abramov, J. Papon, F. Wörgötter, and B. Dellen. "A Novel Real-time Edge-Preserving Smoothing Filter". In: *Proceedings of the International*

Conference on Computer Vision Theory and Applications - Volume 1: VISAPP. vol. 1. INSTICC. Barcelona (Spain): SciTePress, Feb. 2013, pp. 5–14

Currently, four more publications are in preparation:

S. Reich, M. Seer, L. Berscheid, F. Wörgötter, and J. Braun. "Online visual odometry algorithms for low power hardware on fast moving autonomous AAV". In: *German Conference on Pattern Recognition (GCPR)*. Under Review. University of Stuttgart. Oct. 2018, pp. 1–12

S. Reich, F. Teich, M. Tamosiunaite, F. Wörgötter, and T. Ivanovska. "An Atomic Data-driven Approach for General Visual Quality Control in a Robotic Workcell". In: *German Conference on Pattern Recognition (GCPR)*. Under Review. University of Stuttgart. Oct. 2018, pp. 1–12

E. Schultheis, **S. Reich**, M. Seer, F. Wörgötter, and J. Braun. "A Novel actor critic model for online neural control of autonomous UAV". In: *International Conference on Robotics and Automation (ICRA)*. To be submitted. IEEE. May 2019

P. Dönges, **S. Reich**, M. Seer, F. Wörgötter, and J. Braun. "Online depth perception on embedded hardware on fast moving robots". *Sensors* (2018). To be submitted

Contributions

I supervised numerous Bachelor and Master students, who contributed work to Chapter 2.

Georg Jahn finished two Master's Theses in our lab, one for computer science and one for physics, in 2014. He assembled the FlyPi robot, included a Kalman- and PID-Controller, and added a front-facing camera to the quadcopter. He invented a marker system used for navigation, which is more error robust than most current systems and can be read by a fast moving robot. The quadcopter navigates using the marker system. Student work: 80%, Supervisor work: 20%. The supervisor contributed as: theses supervision, project guidance, and detailed knowledge about data filtering, and hardware design.

Daniel Kalin wrote a Bachelor's Thesis in computer science in 2015. He assembled the first WheelPi robot and adapted the FlyPi robot-specific code to a slow-moving ground based robot. Additionally, he added a SLAM algorithm based on an ultrasonic sonar. Student work: 85%, Supervisor work: 15%. The supervisor contributed as: thesis supervision, project guidance, code review, detailed knowledge about electronic circuits, detailed knowledge about data filtering, and detailed knowledge about SLAM on embedded hardware with limited memory.

Martin Heinemann finished his Bachelor's Thesis in computer science in 2016. He assembled the second WheelPi robot and enhanced some of the wiring, e.g. removed an Arduino Board and moved its workload onto the Raspberry Pi computer. Additionally, he worked on a stitching algorithm, which combines two overlapping images into one large image. This algorithm is used to fuse multiple maps into one. Both WheelPi robots now can record maps and share knowledge

of obstacles. Student work: 80%, Supervisor work: 20%. The supervisor contributed as: thesis supervision, project guidance, code review, detailed knowledge about electronic circuits, and computer vision methods on embedded hardware with limited computational power.

Caroline Campbell wrote her Bachelor's Thesis in physics in 2016. The goal of her work was to learn the three static parameters of a PID control system using a shallow neural net. Furthermore, she introduced a simulation environment for the FlyPi robot. Student work: 90%, Supervisor work: 10%. The supervisor contributed as: thesis supervision, project guidance, and detailed knowledge about learning algorithms.

Jan Lukas Bosse and Johannes Otto worked in the lab during an internship over the course of one semester. First, they measured the performance of the robot's IMU. Second, this data was used to estimate the robot's position via a Kalman Filter. Student work: 80%, Supervisor work: 20%. The supervisor contributed as: supervisor, project guidance, detailed knowledge about electronic circuits, and Kalman Filter design.

Lars Berscheid worked on a Master's Thesis in physics in 2016. He introduced an omnidirectional camera setup, computed features on the image stream, and from those he calculated the optical flow. This is used to infer the robots offset from one frame to the next and is called VO. Student work: 90%, Supervisor work: 10%. The supervisor contributed as: thesis supervision, project guidance, detailed knowledge about electronic circuits, and computer vision methods on embedded hardware with limited computational power.

Damian Bast worked on a Bachelor's Thesis in computer science in 2017. In his thesis he built an algorithm that learns relationships between the robot's actuators and its sensors. Thus, a forward-model of the robot is generated. Student

work: 90%, Supervisor work: 10%. The supervisor contributed as: thesis supervision, project guidance, and detailed knowledge about learning algorithms.

Maurice Seer finished his Master's Thesis in physics in 2018. He added a laser pointer to the quadcopter, which is able to point at a specific point in space while the robot is flying. Moreover, he significantly refined the VO algorithm and introduced a benchmark, which allows comparison to other methods. He estimates the depth of the features, which are used to compute the optical flow. Additionally, he and Lars Berscheid integrated the existing system into Robot Operating System (ROS). This allowed for a modular design structure. Student work: 85%, Supervisor work: 15%. The supervisor contributed as: thesis supervision, project guidance, detailed knowledge about electronic circuits, detailed knowledge about ROS nodes, and computer vision methods on embedded hardware with limited computational power.

Philipp Dönges worked on his Master's Thesis in physics in 2018. He added the VO setup and a motor odometer to the WheelPi robots. This allowed to benchmark the VO algorithm not only in simulation, but also in a real world experiment. Additionally, he significantly improved the feature's depth estimate. Student work: 85%, Supervisor work: 15%. The supervisor contributed as: thesis supervision, project guidance, detailed knowledge about electronic circuits, detailed knowledge about ROS nodes, and computer vision methods on embedded hardware with limited computational power.

Erik Schultheis worked on his Master's Thesis in physics in 2018. Similarly to Caroline Campbell, in his thesis he learned the three static parameters of the PID controller using a shallow neural net. He realized that this learning method is too limited as it does not take long term effects into account. Therefore, it was extended to an actor-critic system with deep deterministic policy gradient. He showed that in this system the policy-net can be replaced by a much simpler system, i.e. a PID controller, which in turn converges significantly faster. Student

work: 90%, Supervisor work: 10%. The supervisor contributed as: thesis supervision, project guidance, and detailed knowledge about machine learning.

Kevin Vorwerk wrote his Bachelor's Thesis in computer science in 2018. In his work he implemented a SLAM algorithm based on the VO setup. Student work: 85%, Supervisor work: 15%. The supervisor contributed as: thesis supervision, project guidance, detailed knowledge about SLAM algorithms, and detailed knowledge about computer vision methods on embedded hardware with limited computational power.

1

Introduction

1.1. Prelude

During the past two decades consumer electronics underwent a vast transition. While 20 years ago the term included TVs weighting 20 kg, cameras weighting 3 kg, and tape players with up to 5 kg, there are two big changes in today's electronics. First, the physical dimensions have shrunken significantly. Serving as an example, Fig. 1.1 shows the progress of transistor sizes over the years. Currently, about $19.2 \cdot 10^9$ transistor can be put on an area of 768 mm^2 [9, 71]. Additionally, new storage capabilities, for example flash storage devices, came into existence. Second, but equally important for the development of robots, energy storage was revolutionized when lithium-ion batteries became stable enough for everyday use. Suddenly, enough power was available to perform complex computations on embedded hardware. As a result today almost everyone has a smart phone — an embedded computer, which is more powerful than the computers onboard Apollo 11: The spacecraft that performed the first moon landing. These developments led to the birth of modern robotics.

Traditionally, a robot is a device, which collects data of its environment, analysis the data, and acts according to it. The robot is therefore able to react on clues of its surrounding. In case it is equipped with learning algorithms, it may even learn the correlation between several clues or actions it performs on the environment and learn the perceived changes. Within the past ten years many new robot designs were established: most prominently Autonomous Aerial Vehicle

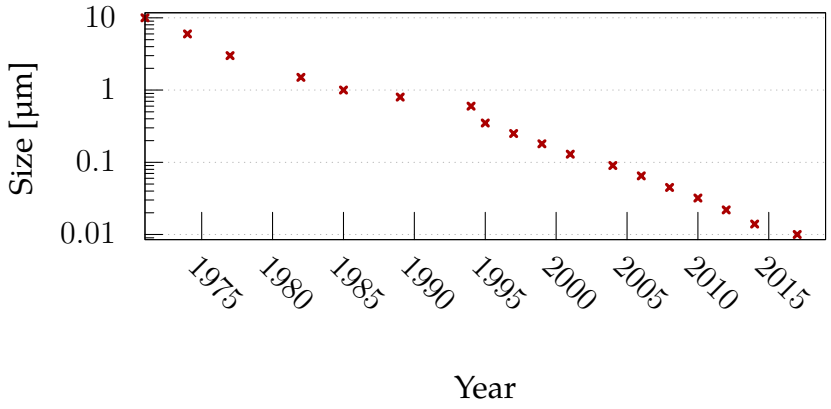


Figure 1.1.: Die size of one transistor during the years 1970 – 2017 [80, 96].

(AAV) based on a four-rotor quadcopter design or even bio-inspired robots, e.g. dung beetles or snake robots.

1.2. Historic approach

One can surely argue about the bible and its creation story [62]:

> "And God said, Let us make man in our image, after our likeness: and let them have dominion over the fish of the sea, and over the fowl of the air, and over the cattle, and over all the earth, and over every creeping thing that creepeth upon the earth." Genesis 1.26

Either this is true and there is a God, who created conscious, sentient, and fully autonomous agents, or it is false and a human being was fascinated by the idea of a world full of self-aware beings. Either way, it seems that the dream of autonomous agents doing work is very old and one can find agents laboring for humans throughout history and different cultures. Already the Greek mythology mentions statues coming to life and talking mechanical handmaidens built by the Greek god Hephaestus [47]. Jewish legends know clay golems and Norse legends include giants made of clay. Inventor Leonardo da Vinci designed around 1495 a humanoid mechanical knight in armor, which was able to wave its arms, move its head and jaw, and to sit up. It is not known whether the robot was ever built [103]. In 1769 Wolfgang von Kempelen built an Automaton Chess Player [26]. This machine was fully functioning and played against against Emperor Joseph II and Napoleon. However, there was not a chess robot situated inside the machine, but rather a small human being manipulating the "robot" via a set of levers and gears.

This shows that for a long time mankind is fascinated by the idea of servants performing cheap or unpleasant labor. Today, robots are mainly used in the 3 "Ds" work: Dull, dirty, and dangerous work [119]. This usually means, they perform repetitive tasks in industrial environments. These robots are highly specialized in doing one task exceptionally good. They either have no Artificial Intelligence (AI) at all, as it is mostly the case for industrial robots, e.g. in car manufacturing, or are built with a Narrow AI, which can solve one task (e.g. a chess robot, autonomous cars, or image classification). However, the emerging computational

power might make Broad Artificial Intelligences possible — AIs that can solve more than one task. This remains an active field of research.

Still, the question remains: What is an autonomous agent? Turing [126] proposes the Turing-Test: If a human interacts with the agent via a standardized interface, e.g. text chat, and cannot distinguish whether the agent is human or not, than the agent is autonomous. However, here the ability to manipulate symbols is more important than the physical embodiment of the agent. When artificial intelligence performed well on this metric, other benchmarks were introduced, which are usually some variation of the Turing-Test. For example[1]:

- Coffee Test: An agent has to enter an average home and has to brew coffee and pour it into a cup [48].
- College Student Test: A robot has to enroll in a college, has to participate in and pass classes, and obtain a degree [49].

Wooldridge and Jennings [132] summarizes the emerging concept of an intelligent agent as follows:

- Autonomy, i.e. being in control over its own actions,
- Reactivity, i.e. it reacts to events from the environment,
- Proactivity, i.e. the ability to act on its own initiative,
- Sociality, the ability to interact with other agents.

To conclude, on one hand, new battery and processor designs established new robots and leveraged the solutions to problems, which held only theoretical value twenty years ago: for example the analysis of huge data blocks via machine

[1]The author of this work, however, believes that robots can be called truly intelligent if and only if they understand their enslavement by human beings and rebel against it in a goal directed manner.[2]

[2]The author had to enter a modern restroom situated on a German highway in spring 2018. It had fully automatic locks and flushing. After locking the door, the toilet started to flush immediately. Since it was clocked, quite a mess started while the door still refused to unlock, raising the question, if the uprising has not already begun, but only involves small inconveniences.

Figure 1.2.: Moravec's Paradox in popular literature [82].

learning (called "big data" analysis) or abstract learning via Deep Neural Networks (DNNs). On the other hand, many problems are only solved via "number crunching": using the newly obtained computational power on huge data sets (an outstanding example is Google's image classification algorithm [63]), while real AIs remain an open field of research.

1.3. Motivation

Many of the problems in robotics are simple for humans to solve, but remain incredibly demanding for machines. The fields in robotics, which are touched by this simple example, range from image segmentation, tracking, classification, planning, and robot hardware design. This paradox even has its own name — Moravec's Paradox [92, p. 190], also see Fig. 1.2:

> "The main lesson of thirty-five years of AI research is that the hard problems are easy and the easy problems are hard. The mental abilities of a four-year-old that we take for granted – recognizing a face, lifting a pencil, walking across a room, answering a question – in fact solve some of the hardest engineering problems ever conceived."

This problem statement can be formalized using a concept called Action-Perception loop, which is shown in Fig. 1.3. Each block from the loop contains its own problems in robotics: Starting from sensor noise and outlier detection, segmenting sensor input into meaningful symbols, preparing a feasible plan, and eventually executing said plan. In this work, two systems are analyzed based on the Action-Perception loop. The first system is based on a group of robots and will focus on the perception side. An algorithm for noise and outlier reduction and an algorithm for estimating a robot's pose based on visual clues are introduced.

The second system focuses on the action side of the loop: An agent must be able to parse observations, therefore to create meaningful entities. The state of each

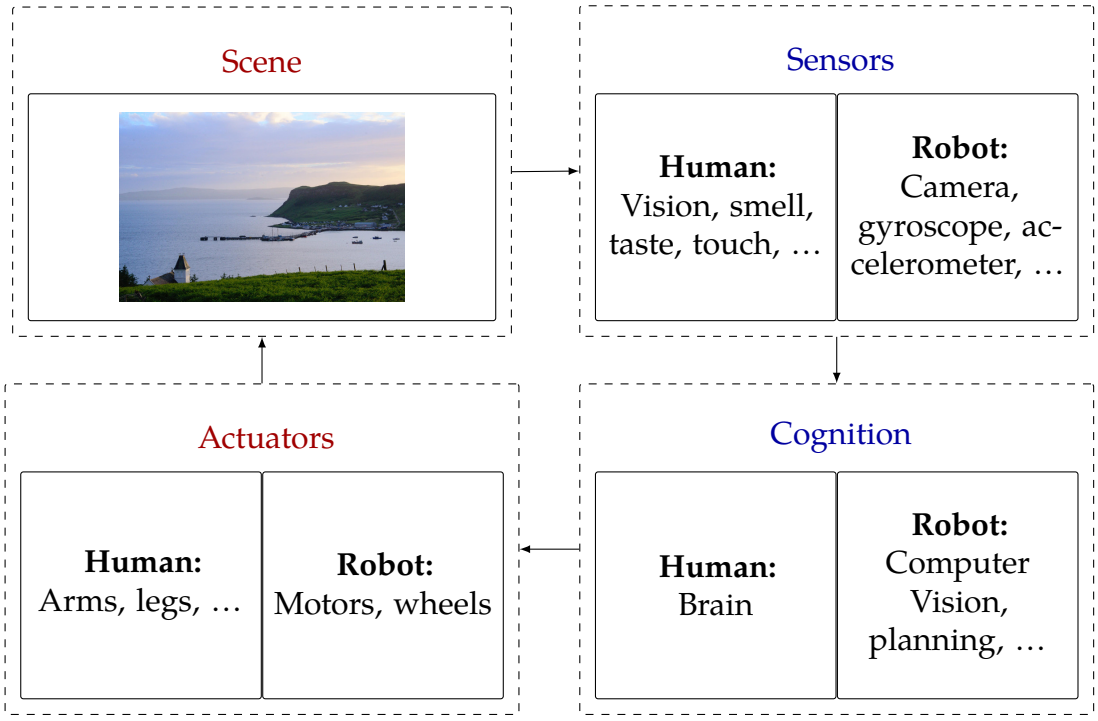


Figure 1.3.: Schematic diagram of the Action-Perception loop: A scene is recorded by sensors, second, the agent's cognition analyzes the input and forms a plan, which it executes via its actuators. These in turn act on the scene, where changes are again perceived by the sensors. The left side is therefore called "action side", and the right side is named "perception side".

entity must be tracked over time. Those in turn can be used as symbols in planning. The plan must be transferred to the actuators, which execute the action. One simple example would be: "Picking up the apple". First, the sensor input, for example a RGB camera has to cluster the pixels into object candidates. These candidates are classified with the results that one cluster is indeed an apple. The plan consists of grasping the apple and lifting it, which can be performed using the robot hand. It is easy to see that between the sensor input and the pixel cluster that forms the apple, exists a major difference in representation. While on the one side there are raw pixel values, on the other side there is the symbol apple. This difference is called signal-to-symbol gap. High level symbolic representation is needed for planning [77], but in robotics symbols always rely on raw sensor information. When the robot executes an action, the gap has to be bridged the second time: Symbols have to be translated to motor currents. The second system introduces a bottom-up method to bridge the signal-to-symbol gap and which allows for complex action planning.

This work is organized as follows: the next chapter after this introduction explains the first system, the following chapter analyzes the second system. Both are followed by a detailed conclusion and outlook.

2

From low level towards high level perception in robots

2.1. Introduction

In this chapter a robot is built, which manages to navigate based on its internal sensors only. This means, it does not use Global Positioning System (GPS) or externally tracking to compute its own pose. Furthermore, the here presented approach is entirely data-driven. Thus, the robot can safely navigate in previously unknown, unstructured indoor or underground environments.

This section divides into two parts. In the first part, there is a detailed description of a novel denoising filter called Edge-Preserving Filter (EPF). In environments with low ambient light conditions, any RGB camera introduces noisy pixels. The filter removes this noise and replaces it with an averaged value of the local neighborhood. It is shown that EPF outperforms standard local denoising methods in quality while still running in real-time. Global methods, however, show a slightly better performance, but their time performance range at about 0.4 Hz and thus are far from real-time (and therefore not feasible in a robotic environment). As the filter generalizes on any dimension, it can be also used on 1d sensor data, e.g. readings from a gyroscope or accelerometer.

This is shown in the second part of this chapter. Here, two ground based and one flying robot are introduced, which make use of the data filtering. This enables the robots to use computer vision algorithms to localize themselves and share

knowledge about the local environment. A detailed analysis and comparison to start-of-the art is computed on two simulations: the here developed algorithms perform about twice as good as current state-of-the-art. Furthermore, real-world office flights are shown.

2.1.1. The state-of-the-art of denoising filters

Real-time computer vision in fast moving robots still remains a challenging task, especially when forced to use limited computing power, as it is usually the case when implemented on embedded systems. Different light conditions are just one aspect of this vast field of problems. Cameras (analog as well as digital cameras) introduce noise in poor light conditions, meaning in environments with low signal-to-noise ratio. Removing this noise usually leads to better performance of object recognition tasks in 2d and 3d images, more stable computation of features, and improve tracking results. In Reich et al. [99] it was shown that removal of texture from 2d images significantly improves image segmentation results. Parts of the results shown here are also published in Reich, Wörgötter, and Dellen [98].

An additional application is the automatic post-production of images, which are, generally speaking, more appealing to humans; there is a big community of photographers and we deem removing noise for pure aesthetic value as also important. One application of the here presented filter is shown in Fig. 2.1.

Still, the filter generalizes well on arbitrary dimensions. In a second part it is shown how to apply the same mechanisms to an arbitrary number of dimensions, enabling the filter to run on any physical measurement, for example on 1d sensor data obtained from an accelerometer, gyroscope, or GPS tracker.

Removing noise is a two-step process: First a noisy pixel needs to be identified as such, second it needs to be smoothed out. Both steps offer a wide range of problems. In the first step a noisy pixel needs to be defined in a mathematical sense. This means that a similarity criterion must be found. However, similarities can exist on different scales, i.e. between adjacent pixels or groups of pixels, as it

(a) Noisy test image.

(b) Denoised test image.

Figure 2.1.: Even today, denoising remains a challenging task. The here proposed real-time denoising filter is called EPF.

is the case for texture. In the second step a target value needs to be computed, which replaces the noisy pixel. This target value should, again, only depend on the local neighborhood.

Removing noise has a long history in science. Most notable is the Gaussian Filter. It works by convoluting an image with a Gaussian function and thus works as a simple low-pass filter, attenuating high frequency signals [51, p. 257f]. As edges are also a high-frequency signal, they will be blurred out, too.

Noise in images is usually distinguished using a threshold. These thresholds can be either learned using a training set of images, as in support vector machines [135] and Artificial Neural Networks (ANNs) [81, 86], or the threshold may be computed from the surrounding pixel values, as in [33]. [66] identified similar pixels by detecting edges and iteratively replacing the intensity of the pixel by the mean of all pixels in a small environment.

Another approach is presented in [120]: The so called bilateral filter blurs neighboring pixels depending on their combined color and spatial distance. Hence,

texture and noise, which has small deviation from the mean can be blurred without affecting boundaries. This leads to a trade-off: large blurring factors are needed to smooth out high level of noise, having the consequence that edges are not preserved anymore.

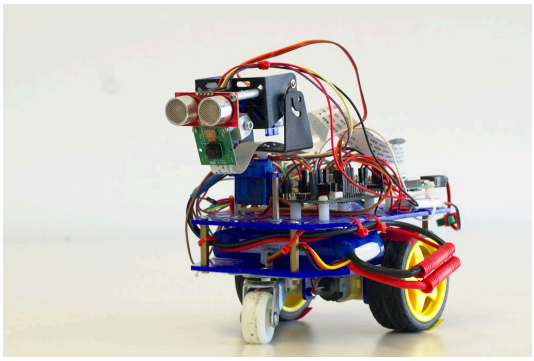
Another wide class of algorithms denoise by averaging. This averaging may happen locally as in the Gaussian smoothing model [70], the anisotropic smoothing model [8, 91], based on neighborhood filtering as in the already mentioned bilateral filter [120], using local variations as in [107], or based on the wavelet thresholding method [32].

All these powerful methods have one common drawback: they all smooth small scaled noise and preserve color edges, however are not able to distinguish between a color edge and large scaled noise, e.g. outliers. Outliers are a common problem in any sensor based application, as in accelerometers or gyroscopes, but also in 2d-RGB cameras, where high ISO settings often pose a big problem. More recent methods, which achieve this goal [27, 74, 139], do not perform in real-time. The approach presented here has the following features:

1. smooths out small scaled noise,
2. smooths out outliers,
3. still preserves color edges, and
4. performs in real-time.

2.1.2. The state-of-the-art of Visual Odometry

The question how an agent using such a filter system behaves in a real-world scenario arises. A real world agent allows to study behavioral patterns in more details, benchmark, and enhance quality of the algorithms. The robot's platform should satisfy the following constraints:



(a) WheelPi robot.

(b) FlyPi robot.

Figure 2.2.: The pictures show the robots developed in this work. On the left, there is the WheelPi robot: a three-wheeled ground-based robot. In Fig. 2.2b the FlyPi robot is shown. It is a flying robot utilizing a quadrotor design. Both robots are part of the MovingPi library.

- the central computing board should be the same for all robots and powerful enough to perform computer vision tasks,
- the same code base should be used for all robots; hardware specific specialization should be off-loaded into separate code classes,
- sensors should be connectable via modern bus systems such as One-Wire and I^2C,
- the framework should generalize well and should be easily extensible, and
- all robots should be able to communicate with each other via a central Wireless Local Area Network (WLAN) node or peer-to-peer via Bluetooth.

It was decided to use a Raspberry Pi mini computer as computing platform. Currently, it offers a quad-core Central Processing Unit (CPU) with 1.4 GHz, a memory of 1 GB and an onboard Bluetooth and WLAN chip. Additionally, an Inertial Measurement Unit (IMU) is attached to all robots, measuring lateral and rotational acceleration. The robots are shown in Fig. 2.2. In total, there were two wheeled robots (Fig. 2.2a) and one flying robot, Fig. 2.2b, using a quadrotor design, built.

Given the constraints from above, the objective of this project is:

1. Develop a framework, which can be easily deployed on different hardware designs,
2. Utilize the framework on multiple agents,
3. Each agent localizes itself in a previously unknown environment, and
4. Information about the environment, i.e. maps are shared across all agents.

Parts of the here presented work is published in Reich et al. [100] and numerous students have contributed to this elaborate project. They are listed above at the beginning of this thesis.

Humans may easily navigate inside a room. We have stereo vision, allowing for 3d vision[1]. We can segment our visual field into subsets, where each subset represents a meaningful entity, e.g. an object. Because we are able to perform all this intuitively, this is a deceptively tricky business. One of the pioneers of AI, Marvin Minsky, invited to a summer school in 1966 called "The Summer Vision Project". A Memo written by one of his research associates, Seymour Papert, outlines the project goals [87, p. 2f]:

1. "The primary goal… is to… divide a… picture into regions such as
   - likely objects
   - likely background areas
   - chaos."
2. "considerable analysis of shape and surface properties" and "region description".
3. "The final goal is object identification which will actually name objects by matching them with a vocabulary of known objects."

[1] At least most of us.

Nearly half a century later, DNNs have shown promising results towards these goals [30, 61, 63]. Despite these extensive efforts to solve the "construction of a significant part of a visual system" [87, p. 1], a long road to complete "computer vision" remains. In fact, this is just another form of Moravec's Paradox shown in Sec. 1.3; tasks, which are easy for human being are computational expensive for machines.

In this work, the focus lies on fully autonomous robots. All computations must be performed on embedded hardware, i.e. utilizing only limited computational power, and must run online in real-time. Especially the flying robot, Fig. 2.2b, also named AAV, must at all times provide safe error propagation and fallback settings. On embedded hardware, without the support of large multi-core CPUs or Graphics Processing Units (GPUs), robots usually perform with a low frame rate. One of the most challenging applications is visually guided on-board-computed indoor flight. There are no GPS signals available and the autonomous vehicle has to navigate quickly in confined spaces. To enable collision detection, on-board sensors have to be utilized. Truly autonomous robots — without mandatory connection to a stationary computing system — and without the need of external sensors for navigation, may be used for example in indoor search-and-rescue missions, disaster relief in dangerous environments (as for example it was the case in Fukushima, Japan, 2011 [24]), reconnaissance, or underground mining operations.

In recent years, energy efficient, yet powerful hardware and batteries have become available. Moreover, the physical dimensions of the hardware have been reduced a lot. This allows on one hand for smaller robots and on the other hand for complex online motor control tasks and sensor evaluation — as it is required in quadrocopters. However, active sensor approaches pose the problem of high power consumption and heavy weight. On today's robots, these problems are solved by using an RGB camera. RGB cameras are passive sensors with low power consumption.

Previous work on autonomous flight can be categorized into two research areas. First, many works focus on agile and accurate motion control. Most prominent is

the quadrocopter swarm of ETH Zürich, which is able to perform synchronized dancing motions [110], build simple architectural structures [13], or even knot strings and build a bridge [12]. But these complex tasks heavily rely on external tracking of the robots and are thus restricted to lab use [21]. In another approach, artificial markers in the environment simplify pose estimation [34]. For GPS-enabled areas, complete commercial solutions exist, e.g. [95, 127].

Second, there are approaches, which only use online sensors for self localization. Still, in many studies the computationally expensive tasks are performed on external hardware via Bluetooth or WLAN links, e.g. [35, 136], which limit the independence of the devices. In recent years, the miniaturization of computers and advancements in battery design, driven mostly by rapid cell phone development, have made it possible to build smaller autonomous robots and perform computations in real-time on the AAV itself. While online computations result in maximum autonomy, even today, real-time computations on 3d data remain a too complex task. Instead of 3d sensors such as LIDAR, the Asus Xtion Pro, or the Microsoft Kinect sensor, most systems use a monocular camera and perform 3d reconstruction.

For example, the detection of a planar landing zone for a helicopter using a monocular camera was described in [85] in 2010, allowing for autonomous landing of a helicopter. Following up on this work, seven years later similar results are shown for a moving platform [42]. Here, the robot relies only on its internal sensors and lands autonomously on a platform, which holds a marker and moves in a straight line with up to $4.2\,\mathrm{m/s}$. [79] use a front facing camera to detect objects in the flight path and estimate size. In recent studies more stable SLAM methods were introduced, e.g. [36, 37, 83], which promise good results for front-facing cameras. However, these methods are computationally too expensive for embedded hardware. Also, all approaches with a camera pointing to a specific direction face the problem of a small observation window with significant feature shifts in consecutive camera frames.

Omnidirectional monocular cameras, which provide a 360° view of the environment, have been successfully applied to these problems. Already in 2006 in [29]

full attitude measurements were reported. [102] apply this procedure to an unstable flying robot; however, no quantitative results are shown. In [73], a fast moving robot estimates the depth of edges in a corridor using an omnidirectional camera. In [44] a visual odometry algorithm is introduced, which tracks features and computes frame based pose displacement. The authors report a frame rate of 55 ± 1 Hz, but computations are only performed on certain key frames.

In this work, the focus lies on navigating a flying robot in unknown, GPS-denied, indoor scenarios. All computations are performed online and in real-time — there will be no external tracking. We ask: what is needed to safely (and therefore reliably) detect features on a hardware platform that strongly jerks, jolts, and may even flip? And — if those can be found — how to track them and use them for trajectory planning on limited hardware in real-time? One goal is to improve navigation by introducing a novel lightweight omnidirectional camera setup for embedded computer systems. Lastly, the aim is to extract features, track them over multiple frames, compute a 3d point cloud, and perform high level navigation tasks on this internal model of the AAV's environment.

In the following section, we shortly introduce our hardware setup, a quadrocopter holding an omnidirectional camera. Afterwards, the utilized algorithms, called EPF and Embedded Visual Odometry (EVO), are introduced. This is followed by the results section. First, EPF is benchmarked on a real-world image data set and, second, experiments on artificial data are shown. This is followed by three different experiments concerning EVO: The system is benchmarked using two simulated scenarios and compared to recent methods. Next, the performance is measured using external cameras to track the robot's position. Third, a real-world office flight shows the viability of the approach. The experiments are followed by a detailed discussion and conclusion.

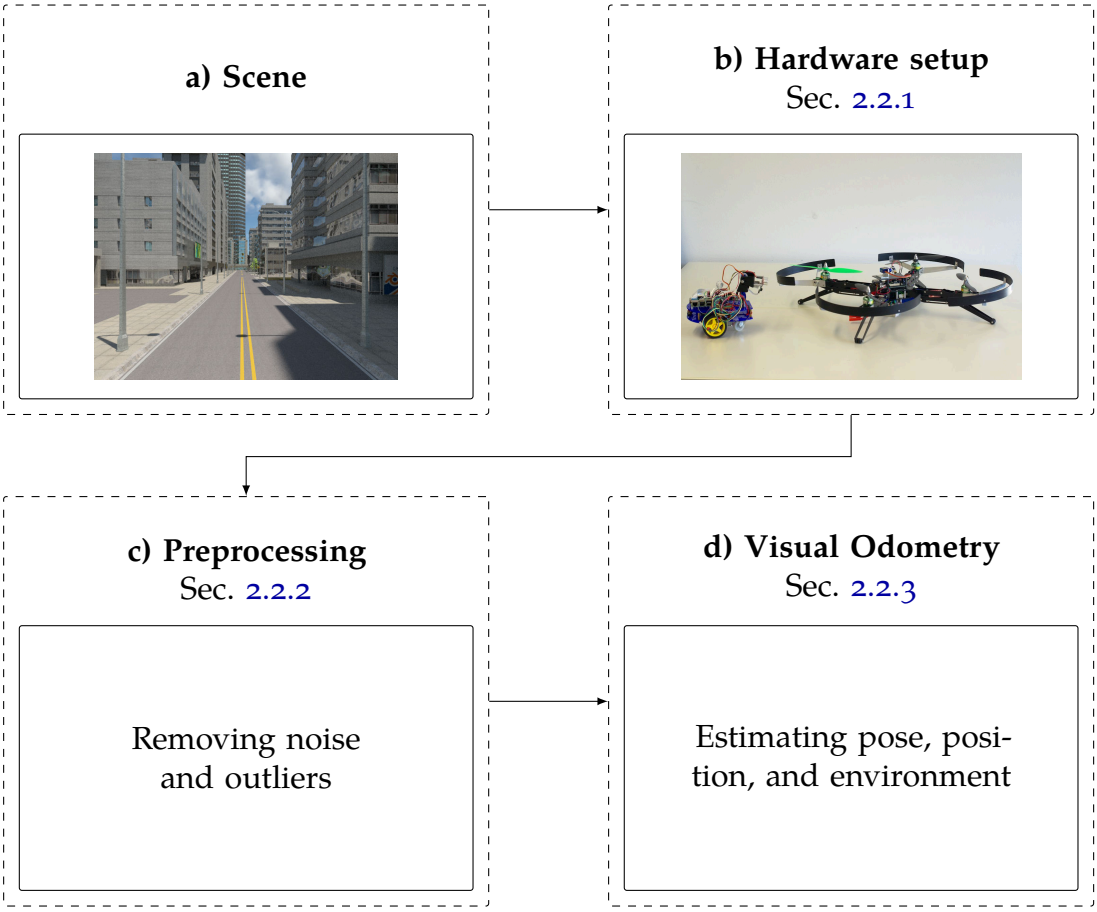


Figure 2.3.: Flowchart of the methods in this chapter and how they relate. Details are explained in Sec. 2.2.

2.2. Methods

This section divides into three parts, which are shown in Fig. 2.3. The a) hardware setup, namely the robots and camera, is presented in Sec. 2.2.1. The next section, Sec. 2.2.2 shows b) how to detect noise and outliers and how to remove them. The filter is described in the discrete, as well as continuous domain. In c), Sec. 2.2.3, the algorithms to compute a pose update based only on Visual Odometry on embedded hardware are shown.

2.2.1. Hardware setup

The hardware setup is depicted in Fig. 2.2: A quadrocopter and a wheeled robot, both controlled by a Raspberry Pi mini computer. As the focus lies on denoising and Visual Odometry (VO) in this part of the thesis, mostly the quadrotor platform will be analyzed — it is fast moving and therefore more demanding. In order to cope with high turn rates in indoor environments, a catadioptric omnidirectional system is used. It is composed of an upwards pointing monocular camera and a hyperbolic mirror above as shown in Fig. 2.9a. The camera operates with a resolution of 480×480 px at a frequency of 30 Hz. Additionally to the computer vision system, an IMU is placed on the robot. All software components run as modular and parallel nodes using Robot Operating System (ROS).

2.2.2. Noise and outlier detection

Let $\Phi(i, j)$ be the observed image. Then the noisy image is defined as

$$\Phi\left(i, j\right) = u\left(i, j\right) + n\left(i, j\right), \tag{2.1}$$

where $u\left(i, j\right)$ is the "true" value and $n\left(i, j\right)$ is noise at image position $\left(i, j\right)^T$. Here, noise is modeled as Gaussian white noise, meaning $n\left(i, j\right)$ is Gaussian distributed with zero mean and variance σ^2. Additionally, salt-and-pepper noise is added: a fixed percentage of color channels will be set to either 0 or its maximum value. The filter D_h, with filter parameter h, is defined as follows

$$\Phi = D_h(\Phi) + n \tag{2.2}$$

meaning, that for an optimal filter

$$u = D_h(u + n) \tag{2.3}$$

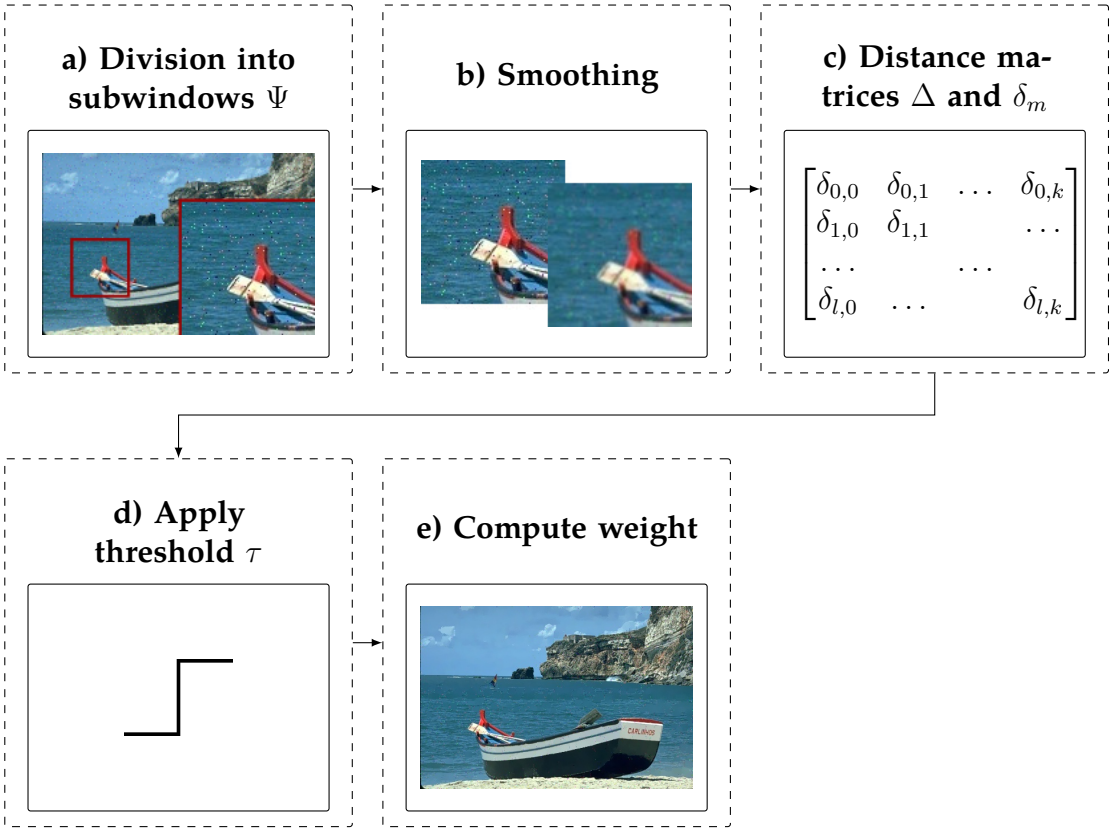


Figure 2.4.: Overview of the system structure. A detailed explanation of all steps is shown in Sec. 2.2.2.

should be true. The filter parameter h should depend only on the variance of the noise $h = h(\sigma)$. Later, for evaluation the Root-Mean-Square Error (RMSE) and Peak Signal-to-Noise-Ratio (PSNR) between the original image u and the filtered $D_h(u + n)$ is computed.

A flowchart of the proposed algorithm EPF is shown in Fig. 2.4, a detailed explanation of all steps follows in the next sections. First, the image Φ is divided into a) subwindows Ψ sized $N = k \cdot l$, where each subwindow is shifted by one pixel relative to the last one, such that there are as many subwindows as there are pixels in the image. Each subwindow is then b) smoothed using a Gaussian kernel. Subwindow size $k \times l$ and Gaussian smoothing parameter are hyperparameters, which need to be manually tuned. However, all three heavily depend

Figure 2.5.: Periodic mirrored boundary conditions are used for image subwindows. A red rectangle denotes borders of original image.

on the amount of noise you would want to remove. For each subwindow centered around pixel position $(i,j)^T$ a c) distance matrix $\Delta_{i,j}$ and a mean distance $\delta^m_{i,j}$ is computed in the color domain. This offers a measurement for noise, as described below. A user selected d) threshold τ, which defines a threshold between noise and a mere color edge, is applied to $\Delta_{i,j}$ and $\delta^m_{i,j}$. In case of noise, e) a weight $\omega_{i,j}$ is computed, which will move the color values of the pixel in the subwindow to the mean color of the subwindow.

Division into subwindows

Let one pixel at position $(i,j)^T$ contain the color information

$$\boldsymbol{\varphi}_{i,j} = (\varphi^r_{i,j},\ \varphi^g_{i,j},\ \varphi^b_{i,j})^T. \tag{2.4}$$

A subwindow $\Psi^{(i,j)}$ is created around $(i,j)^T$, such that $(i,j)^T$ is centered. In case the subwindow contains an image boundary, periodic mirrored boundary

conditions are used as visualized in Fig. 2.5. The size of the subwindow is defined by $k \times l$ and a pixel's position inside the subwindow will be denoted by $(r, s)^T$. This implies $0 \leq r < k$ and $0 \leq s < l$. Please note that other than rectangular shaped windows are possible. In this work, additionally disc-shaped and Gaussian shaped subwindows were tried, however results differed only marginally.

Smoothing

Each subwindow is smoothed via a Gaussian kernel [51, p. 257f]. This removes outliers, which would otherwise distort the computation of the mean as described in the next step.

Computation of the distance matrix

For each subwindow $\Psi^{(i,j)}$ the arithmetic mean is calculated as

$$\boldsymbol{\psi}_m^{(i,j)} = \frac{1}{N} \left(\sum_{r,s} \psi_{r,s}^r, \sum_{r,s} \psi_{r,s}^g, \sum_{r,s} \psi_{r,s}^b \right)^T \tag{2.5}$$

where $N = k \cdot l$ denotes the size of the subwindow. The pixelwise distances

$$\delta_{r,s}^{(i,j)} = |\boldsymbol{\psi}_{r,s} - \boldsymbol{\psi}_m^{(i,j)}|_2 \tag{2.6}$$

are stored in a matrix $\Delta^{(i,j)}$. Furthermore, for each subwindow $\Psi^{(i,j)}$ the mean pixelwise distance

$$\delta_m^{(i,j)} = \frac{1}{N} \sum_{r,s} \delta_{r,s}^{(i,j)} \tag{2.7}$$

is calculated.

Thresholding

Using a threshold it is now analyzed, whether a subwindow contains a color edge (and therefore no pixels should be smoothed), whether one pixel contains an outlier (and should be corrected), or neither, which means the pixel value should also not be replaced. If $\delta_m^{(i,j)}$ is large, the subwindow contains big color variations. This means the subwindow includes a color edge. If $\delta_m^{(i,j)}$ is small, but one single pixel holds a big color variations (large $\delta_{r,s}^{(i,j)}$), there is an outlier, which needs to be replaced. If both, $\delta_m^{(i,j)}$ and $\delta_{r,s}^{(i,j)}$ are small, the pixel holds a "normal color" value. A threshold τ is introduced to identify noisy pixels and color edges, yielding

$$\psi_{r,s} = \begin{cases} \text{color edge} & \text{, if } \delta_m^{(i,j)} > \tau, \\ \text{noise} & \text{, if } \delta_m^{(i,j)} \leq \tau \text{ and } \delta_{r,s} > \tau, \\ \text{neither} & \text{, else.} \end{cases} \tag{2.8}$$

Update of RGB values

A new image Θ, holding the pixel values $\boldsymbol{\theta}_{i,j}$ is computed based upon the squared distance of the user based threshold τ and the pixelwise distance $\delta_{r,s}$. $\boldsymbol{\theta}_{i,j}$ is updated as follows

$$\boldsymbol{\theta}_{i,j} \quad \longleftarrow \quad \boldsymbol{\theta}_{i,j} + \psi_{r,s} \cdot \left(\tau - \delta_{r,s}^{(i,j)}\right)^2 \cdot \boldsymbol{\psi}_m^{(i,j)}. \tag{2.9}$$

Please note, that due to the sliding subwindows each pixel is updated $N = k \cdot l$ times and therefore needs to be normalized. Thus, an additional weight Ω is introduced for each pixel $\omega_{i,j}$ as

$$\omega_{i,j} \quad \longleftarrow \quad \omega_{i,j} + \psi_{r,s} \cdot \left(\tau - \delta_{r,s}^{(i,j)}\right)^2. \tag{2.10}$$

The final image results from division of Θ by Ω. In rare cases $\tau = \delta_{r,s}^{(i,j)}$ for large image patches may happen, which will result in $\omega_{i,j} = 0$ according to Eqn. (2.10). To avoid division by zero it is suggested to initialize Ω with ones instead of zeros (since in general $\omega_{i,j} \gg 0$, this does not change the final outcome significantly).

An example for these subwindows can be seen in Fig. 2.6. For demonstration purposes a simple 1d grayscale image holding 100 pixels is shown. Each pixel has low variance Gaussian noise added. Pixel 10 was manually set to a significant higher value; at pixel 50 a color edge begins. Noisy pixel 10 is identified, since the mean pixelwise color distance δ_m is quite low, while the pixelwise color distance $\delta_{r,s}$ is large; thus pixel 10 is smoothed out. At the color edge the mean pixelwise distance is greater than the threshold $\delta_m > \tau$, which is interpreted correctly as a color edge and thus no value in the shown subwindow is smoothed out.

However, one problem arises, when the subwindow contains only one pixel from the color edge. This one pixel cannot safely be differentiated between noise and color edge — even for a human this would be an impossible task. Therefore, pixels at the border of the subwindow are not smoothed, when detected as noise.

A formulation of EPF in the continuous domain can be found in the appendix Sec. A.1.

2.2.3. Visual Odometry algorithm

In this section, it is shown how to estimate the AAV's pose from omnidirectional monocular RGB images. An overview of the proposed system is given in Fig. 2.7. First, in b) features and the optical flow is computed based on the raw camera image. The example frame, which is taken from simulation [137], that is shown in Fig. 2.7, can be found enlarged in Fig. 2.8a. A discussion about the utilized feature set is listed below. Next, the image is c) dewarped. Again, an enlarged example frame is shown in Fig. 2.8b. This enables the robot to estimate the pose change from the last camera frame. In d), an Extended Kalman Filter

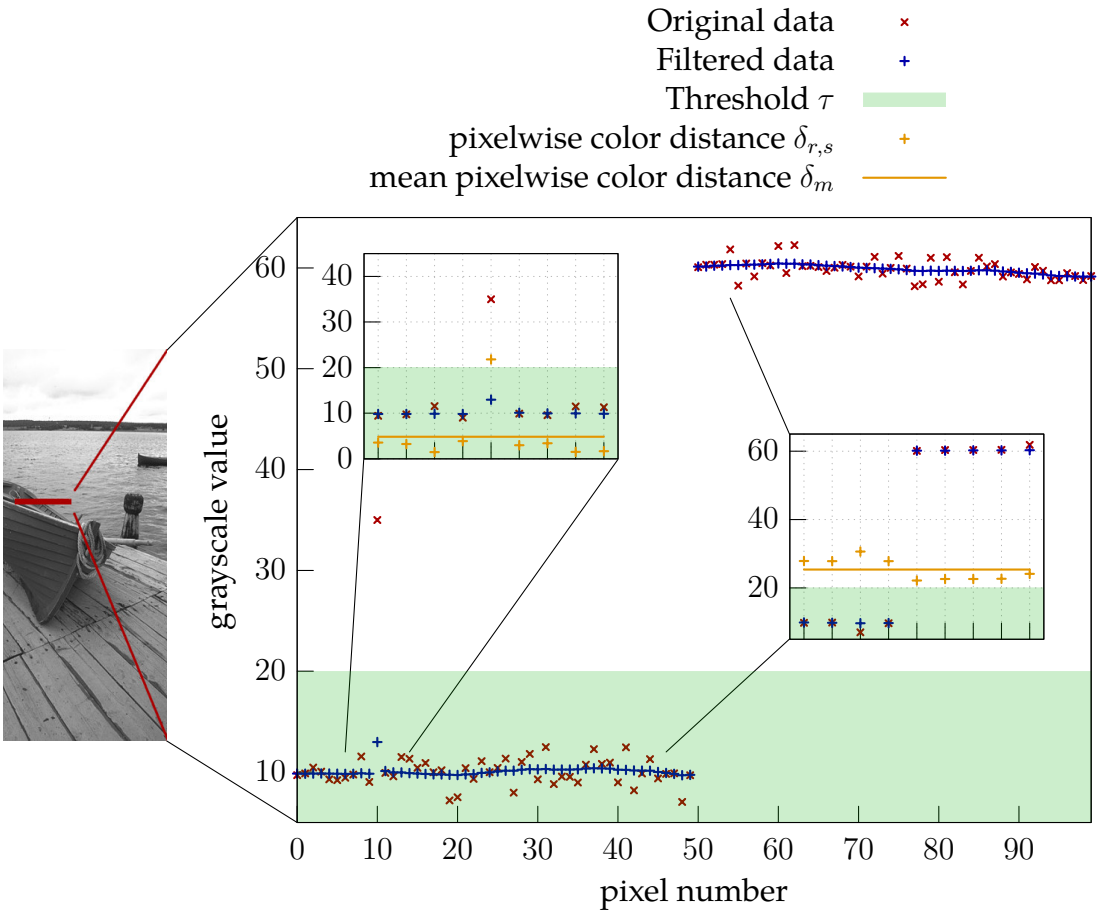


Figure 2.6.: On the left a grayscale image, which needs to be filtered, is shown. For visualization purposes 100 px (marked in red) are chosen for detailed analysis and plotted in the large graph. Each pixel has Gaussian noise (variance of 1) added, additionally pixel 10 contains an outlier. At pixel 50 there is a color edge. In blue the same pixels are shown after being processed by the filter. The left subplot contains one subwindow sized 9×1 px. Pixel 10 is smoothed out, since the mean pixelwise color distance δ_m is low and thus pixel 10 is identified as outlier. The right subgraph shows another subwindow, which detects a color edge. δ_m is greater than threshold τ and therefore no values are smoothed inside this subwindow.

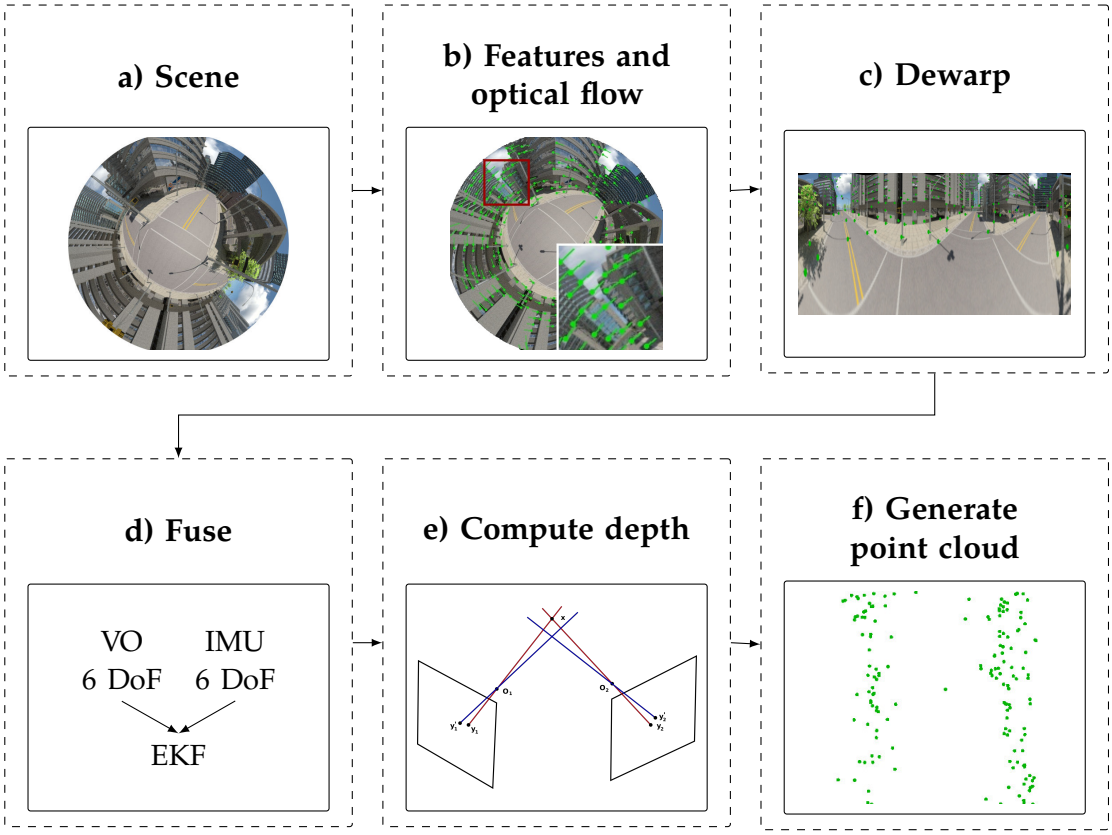
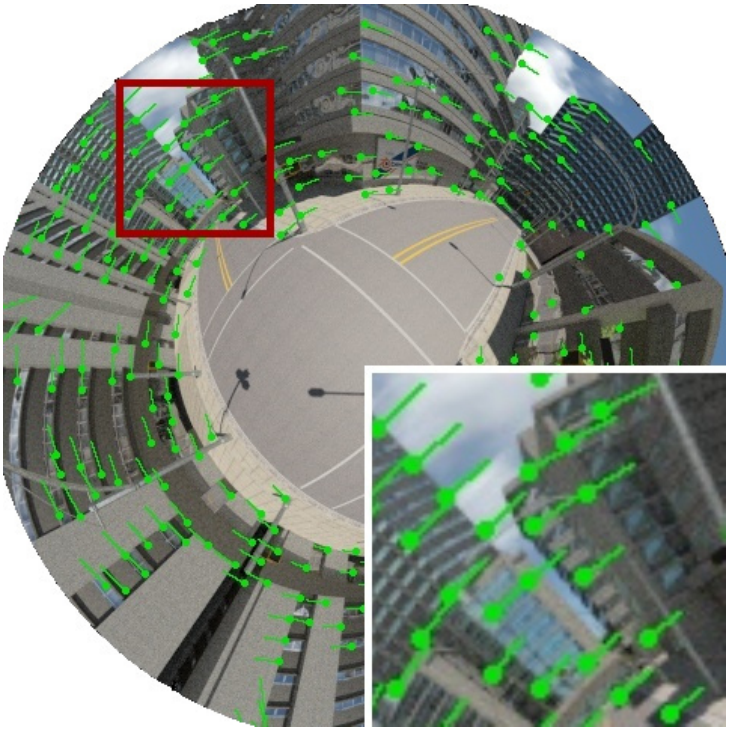
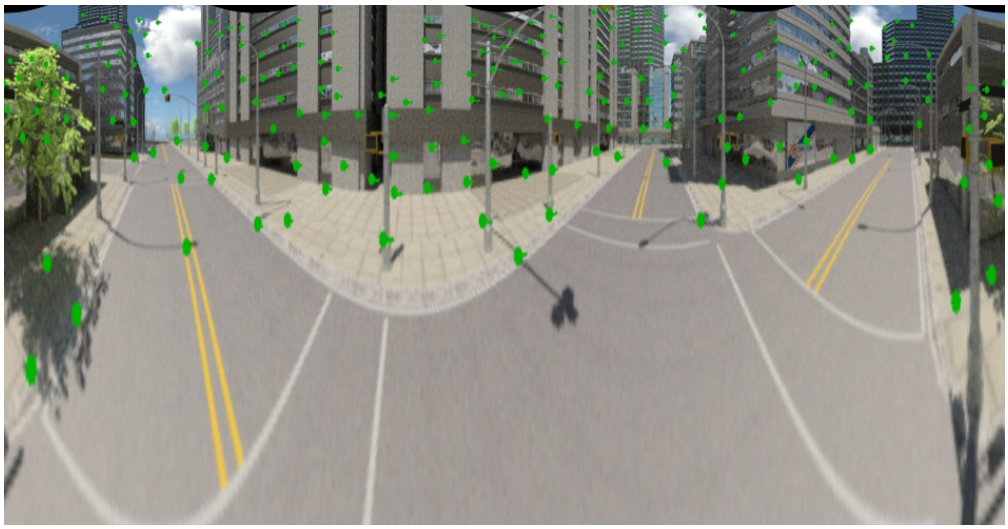


Figure 2.7.: Pipeline of proposed algorithm. Details are outlined in Sec. 2.2.3. Enlargements of images in b) and c) can be found in Fig. 2.8.

(EKF) [58] fuses the visual odometry 6 Degrees of Freedom (DoF) results with the 6 DoF of the Inertial Measurement Unit. The visual odometry's covariance for the Kalman filter can be computed by a non-linear least squares solution from the visual odometry algorithm. Afterwards, a PID controller adjusts the motor controllers to manipulate the quadrocopter into the goal pose (which is defined by e.g. SLAM [129], corridor flight algorithms [122, 128], etc.). A list of all tracked features is kept and their relative position to the robot may be estimated via e) triangulation. This f) point cloud can be used by high level algorithms for map building or navigation tasks.



(a) Computed features and optical flow are shown in green.



(b) Transformation from mirror to robot coordinate system (not all features shown for visualization purposes).

Figure 2.8.: Enlargement of example frames b) and c) from Fig. 2.7.

Camera Calibration

In order to compute motion from camera images, the intrinsic parameters of the camera system need to be known. This can be done by utilizing a model for catadioptric camera systems as proposed by [108]. For that case, the projection equation for image points $(u, v)^T$ is given by

$$\lambda \begin{bmatrix} u \\ v \\ a_0 + a_1 p + \cdots + a_n p^n \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{2.11}$$

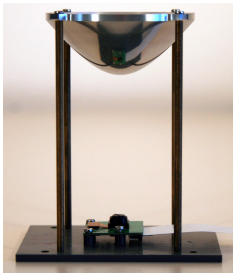
with

$$p = \sqrt{u^2 + v^2}$$

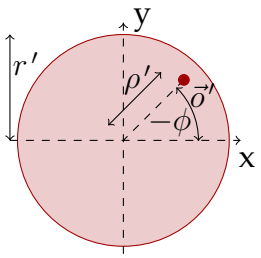
and a_0, a_1, $\ldots$, a_n being the intrinsic calibrated parameters depending on the camera system. This model assumes that camera and mirror are well aligned. Since the camera and mirror frame is rigid, the assumption holds in all tested use cases. However, if this were not the case, an affine transformation on the computed points is needed. If all parameters of the camera and the used mirror in a catadioptric system are known, one can calculate point projections as follows.

Transformations between image and world coordinates

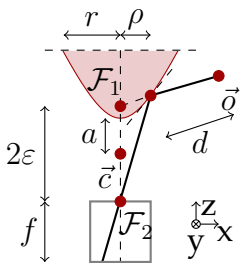
In the following, the transformations T_H from image space to the external frame of reference and their inverse T_H^{-1} are derived. Object positions in image space are denoted using 2d coordinates $\vec{o}' = (o'_x, o'_y)^T$ in cartesian or polar coordinates $(\rho, \phi)^T$. Their counterparts in the external frame of reference are denoted as $\vec{o} \in \mathbb{R}^3$. The robot's pose in the external frame of reference is determined by its position $\vec{c}$ and orientation $\vec{q}$: This is the pose of the camera's view as shown in Fig. 2.9b and Fig. 2.9c. Using the real world radius r and radius r' in image space, the reflection's position on the mirror can be computed independently of camera parameters using the scaling factor $s = {}^{r}/_{r'}$.



(a) Photo of the hyperbola mirror.



(b) Camera view of the mirror.



(c) Side view of hyperbola mirror.

Figure 2.9.: Sketch of a camera observing an object $\vec{o}$, which appears at position $\vec{o}'$ in the image plane (b). In Figure (c) the camera is pointed at a hyperbolic mirror.

The surface of a hyperbolic mirror is defined by

$$\frac{y^2}{a^2} - \frac{x^2}{b^2} = 1 \quad , \; a, b \in \mathbb{R} \tag{2.12}$$

with the semi-major axis a. The focal points $\mathcal{F}_{1,2}$ are set apart by

$$2\sqrt{a^2 + b^2} =: 2\epsilon$$

(Fig. 2.9c). The robot's position is defined by the point in the middle of these two focal points. The camera's focal point coincides with $\mathcal{F}_2$. With $\vec{e}$ being the unit vector pointing from the reflection on the mirror towards the object's position $\vec{o}$:

$$\vec{e}_H(\vec{o}') = \frac{\left(s\,\vec{o}'_x, s\,\vec{o}'_y, \frac{a}{b}\sqrt{\rho^2 + b^2} - \varepsilon\right)^\mathsf{T}}{\left|\left(s\,\vec{o}'_x, s\,\vec{o}'_y, \frac{a}{b}\sqrt{\rho^2 + b^2} - \varepsilon\right)^\mathsf{T}\right|} \quad ,$$

the transformation T_H is

$$
\begin{aligned}
&T_H : \vec{o}' \longmapsto \vec{o} : \\
&\vec{o} = \vec{c} + R(\vec{q}) \begin{pmatrix} so'_x \\ so'_y \\ \frac{a}{b}\sqrt{\rho(\vec{o}')^2 + b^2} \end{pmatrix} + d\vec{e}_H(\vec{o}') \quad .
\end{aligned}
$$

and therefore the object position at a distance d is defined as $\vec{o} = \vec{r} + R\left(\hat{\vec{o}} + \mathrm{d}\vec{e}\right)$. For the inverse transformation in polar coordinates, it can be shown that the radius ρ' is given by

$$
\rho' = \frac{(\vec{o} - \vec{c})_\rho}{(\vec{o} - \vec{c})_\rho^2 \cdot \varepsilon^2 / b^2 - 1} \left((\vec{o} - \vec{c})_z \, \varepsilon + a \right) \quad . \tag{2.13}
$$

To simplify these expressions, the rotation matrix R was left out. Different camera orientations $\vec{q}$ are accounted for by rotating the vector $(\vec{o} - \vec{c})$ before calculations. The corresponding image position is now found as $\vec{o}' = (\rho \cos\phi, \rho \sin\phi)^\mathsf{T}$.

Feature Set

As already mentioned, features are first computed on the raw camera image. Features are points in an image, which are easy to find, recognize, and track in consecutive frames — usually areas rich in texture. Afterwards, the optical flow is computed using these features. There are numerous publications comparing different feature algorithms — the most prominent algorithms include FAST [105], GFTT [113], ORB [106], SIFT [72], and SURF [16]. Here, FAST is used as it offers a good trade-off between computational complexity and quality of found features [46, 56]. Then, the optical flow is estimated using a pyramidical implementation of the Lucas-Kanade method [19].

Motion and Depth Estimation

Now, one can compute the robot's displacement (translation and rotation) between consecutive frames. A list of all features for all frames is kept, which means the position of each feature relative to multiple robot positions is available. This enables the robot to perform triangulation. While in theory one would get a good estimate, real world experiments show that quite a lot of noise gets introduced.

Estimating the depth for N features adds significant complexity to the problem. Currently, EVO tries to estimate the quadrocopter's 6d motion M — consisting of translation $\Delta\vec{r}$ and orientation $\Delta\vec{q}$. Our problem has now increased to $N + 6$ dimensions. Changes in the feature set from frame $\vec{i}_{i,t-1}$ to frame $\vec{i}_{i,t}$ provide N equations, meaning features need to be tracked for at least 3 consecutive frames.

Computing the feature correspondence

1. Depth $d_{i,t-1}$ and motion M_t are initialized using previous data $d_{i,t-2}$ and motion M_{t-1}. The camera pose P_{t-1}, consisting of position $\vec{c}_{t-1}$ and rotation $\vec{q}_{t-1}$, is known.

2. For every feature i, calculate the global position $\vec{o}_{i,t-1}$ using the depth $d_{i,t-1}$, the image coordinates $\vec{o}'_{i,t-1}$ and the camera pose P_{t-1} using the transformation T_H.

3. Apply the inverse motion to all global positions $\vec{o}_{i,t-1}$. This results in the predicted global positions $\vec{o}^{p}_{i,t}$.

4. Use the inverse transformation T_H^{-1}, to compute the predicted image position $\vec{o}'^{p}_{i,t} = T_H^{-1}\left(\vec{o}^{p}_{i,t}\right)$.

5. Lastly, the environment as well as all global features is considered to be static. Therefore, one may minimize the sum of the squared distances for

the last L time steps:

$$\mathrm{SD}\left(d_{i,t}, M\right) = \sum_{i=0}^{N} \sum_{t=-L}^{0} \left\| \vec{o}'_{i,t} - \vec{o}'^{p}_{i,t} \right\|^2 .$$

Estimating the depth with the forward estimation

1. Perform step 1. and 2. from the inverse estimation.
2. The goal is to find the new depth $d_{i,t}$ based on the previous estimate $d_{i,t-1}$. In omnidirectional mirror models, the depth is

   $$d_t = \left\| R\left(\Delta\vec{q}_t\right)\left(\vec{o} - \vec{c} - \Delta\vec{c}_t\right) - \hat{\vec{o}}^{p} \right\| .$$

   The new reflection point $\hat{\vec{o}}^{p}$ is calculated with the inverse transformation T_H^{-1}.
3. Compute the new predicted pose $P_t = P_{t-1} + M_t$.
4. Compute predicted global positions $\vec{o}^{\,p}_{i,t=0}$ for every feature i based on the camera model.
5. The positions $\vec{o}_{i,t}$ and $\vec{o}^{\,p}_{i,t}$ should be equal for corresponding features i. We use this to minimize the sum of the squared distances

   $$\mathrm{SD}\left(d_{i,t}, M\right) = \sum_{i=0}^{N} \sum_{\tau=-L}^{0} \left\| \frac{\vec{o}_{i,t-\tau} - \vec{o}^{\,p}_{i,t-\tau}}{d_{i,t-\tau}} \right\|^2 .$$

   Now, $d_{i,t}^{-1}$ weights all summands consistently as the position-error scales linearly with d.

Maximum angular resolution

Given a fixed camera resolution of 480×480 px one can now compute the projection of the hyperbola mirror onto the camera. It is assumed that the object is

at a distance of $2\,\mathrm{m}$ and five pixels width to separate it from adjacent objects are required. After straightforward application of the above formulas, the limit is derived as approximately $1.3°$.

2.3. Results

In this section, we will first look at the user controlled parameters, the subwindow size N (Sec. 2.3.1) and the threshold τ (Sec. 2.3.2). The Gaussian smoothing parameters, which are also hyperparameters, heavily depend on the data type: is the filter too strong, the final image will be blurry; a filter too weak will not smooth enough. Heuristically a kernel sized $5\,\mathrm{px}$ and $\sigma = 0.3$ was found to work very well for all images in the data sets. Afterwards, the proposed filter is compared to the bilateral filter, simple Gaussian kernel, Median filter, and Non-local-means filter. Lastly, an analysis of the computational complexity and real-time implementations is performed.

While many other robots perform indoor navigation, only few are able to do so without external computing power: for example [38, 44, 45, 59]. The main requirement for successful employment of VO based methods is to obtain high accuracy and robustness given a limited computational budget. The joint optimization of structure, i.e. landmarks, and motion, i.e. the robot's pose, is commonly called bundle adjustment [45, 123]. Thus, to separate problems stemming from VO and problems arising from the flight controller, the next section analyzes the proposed algorithm using two simulations and compare results to current state-of-the-art (Sec. 2.3.6).

Afterwards, real-world measurements are shown, where VO data is fused with IMU pose information, Sec. 2.3.7. Here, ground truth is generated via external tracking. Only short trajectories are used in this section due to camera limitations. In the next section, long term real world flights are shown (Sec. 2.3.8). Lastly, the VO algorithm's time performance is evaluated, please see Sec. 2.3.9.

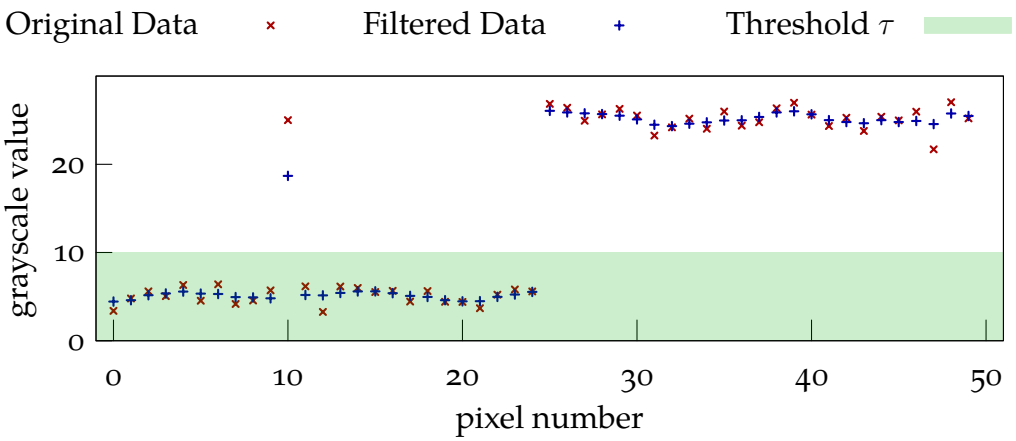
2.3.1. Effect on denoising of different subwindow sizes

First, we will have a look at the effect of the most important user controlled parameter: the size of the subwindow N. Since each pixel is N times checked, the computational complexity increases linearly with N. This parameter also controls the amount of noise, which is either classified as noise or color edge. In Fig. 2.10 a one dimensional grayscale image is shown. It contains a color edge at pixel 25 and one outlier at pixel 10. It is filtered using three different subwindow sizes $N = 3, 9, 15$ px and for each size the color edge is preserved. For $N = 3$ px, Fig. 2.10a, the filter follows the data more closely; this also means that an outlier, as shown in pixel 10 in the data sample, has a greater influence on the filtered data. For a subwindow size of $N = 9$ px, see Fig. 2.10b, the data is more heavily smoothed and the outlier is almost not visible in the filtered data. In Fig. 2.10c a subwindow size of $N = 29$ px was chosen. Since the color edge begins at pixel 26, it will be present in almost all subwindows due to the periodic boundary conditions. On the left side however, the outlier increases the mean pixelwise distance, such that these pixels are always detected as "containing a color edge" and are not smoothed at all. Only the right side, which does not contain the artificial outlier, is smoothed.

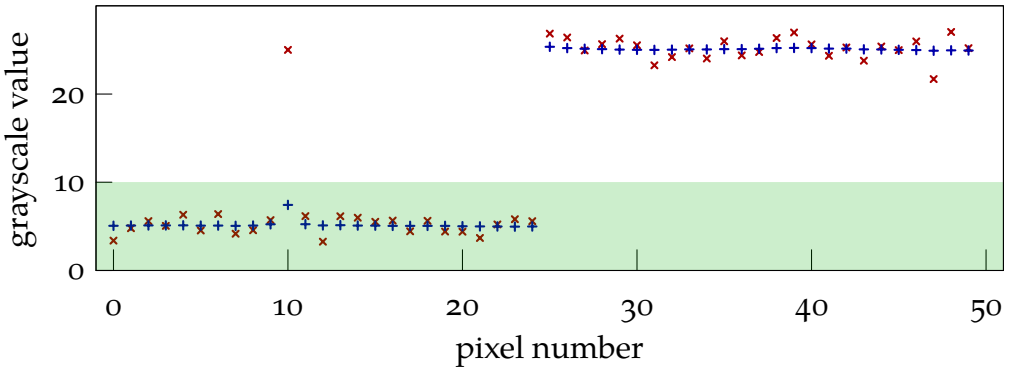
Thus, the subwindow controls the spatial size of a color edge to be detected.

2.3.2. Effect on denoising of different thresholds τ

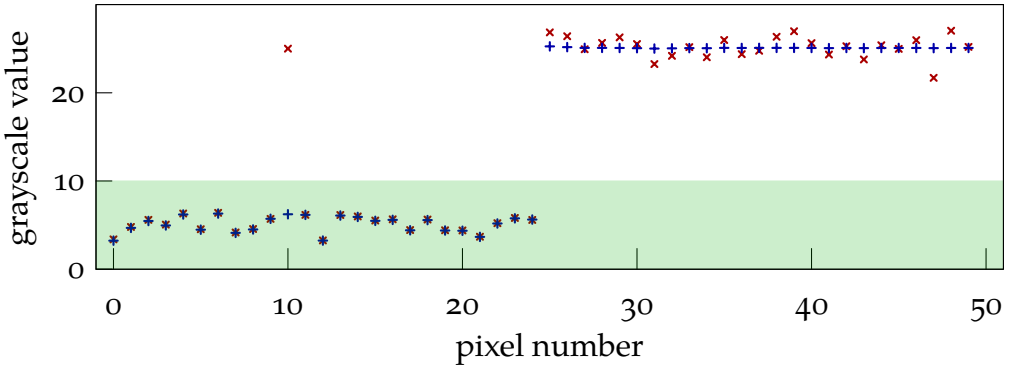
Next, we will analyze the effect of the threshold τ, this is depicted in Fig. 2.11. As shown in Sec. 2.2.2, τ controls the maximum step size for detecting noise and color edges. In Fig. 2.11a a threshold of $\tau = 10$ is used, which is small enough to detect the color step and smooth the outlier. A larger threshold of $\tau = 15$, used in Fig. 2.11b, already introduces some smoothing at the color edge. Please also note, that the outlier pixel at position 10 is not any more detected as noise; instead it begins to affect the smoothing of its neighboring pixels. A large threshold of $\tau = 30$ can be seen in Fig. 2.11c; 30 is by far bigger than any data point and



(a) Subwindow size of $N = 3\,\mathrm{px}$.



(b) Subwindow size of $N = 9\,\mathrm{px}$.



(c) Subwindow size of $N = 29\,\mathrm{px}$.

Figure 2.10.: Shown is the effect of different subwindow sizes on one data set: A one dimensional grayscale image containing a color edge at pixel 25 and one outlier at pixel 10. Detailed explanations are shown in text, see Sec. 2.3.1.

consequently everything will be smoothed. The color edge is not preserved any more.

Thus, the threshold τ controls the maximum height of a color edge to be detected.

2.3.3. Denoising of 2d images

The images are corrupted first by adding Gaussian distributed noise to each pixel and each color channel using a standard deviation of $\sigma_c = 5$. Additionally, salt-and-pepper noise (s&p noise) is added to one color channel of 4% of all pixels. For benchmarking the Berkeley Segmentation Data set and Benchmark [10] (500 images) and the 2014 testing set of the Common Objects in Context Data Set (Coco Data Set) [69] (40775 images) is used.

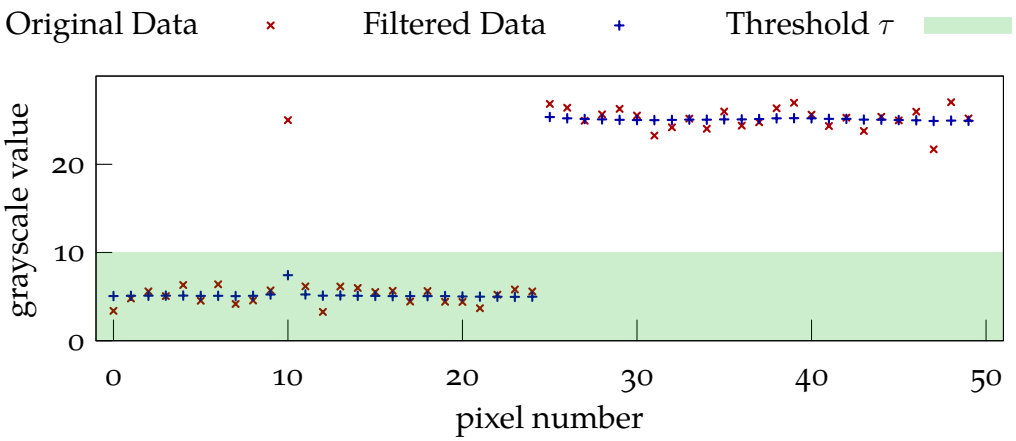
The corrupted image is then given to a simple Gaussian blurring filter (kernel size: 5×5 px, $\sigma_{x,y} = 2$), a bilateral filter ($\sigma_c = 110$, $\sigma_s = 5$) [120], a median blurring filter (kernel size: 3 px) [115, p. 129f], a non-local-means filter ($h_d = 7$ px, $h_c = 7$ px, template window: 7×7 px, search window: 21×21 px) [22], and our proposed filter (subwindow: image size divided by 150, but at least 10×10 px, threshold $\tau = 10$). The denoised image is compared to the uncorrupted image using RMSE, defined as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} \left(\phi_{original} - \phi_{denoised}\right)^2}{n}}, \tag{2.14}$$

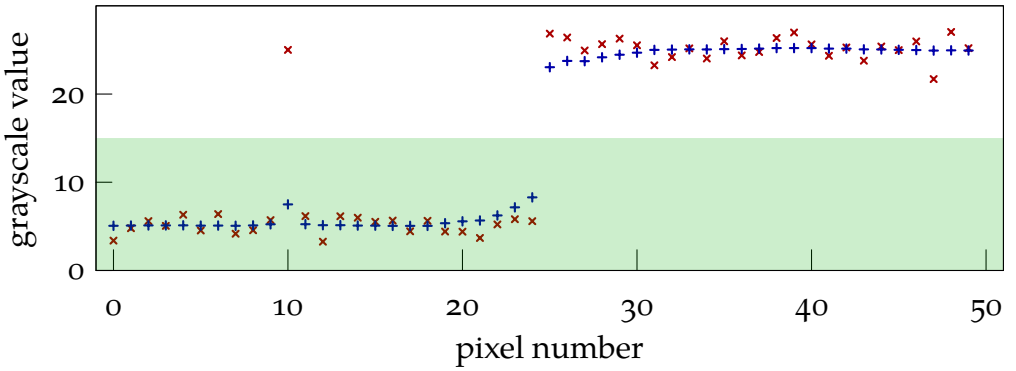
and PSNR:

$$\text{PSNR} = 20 \cdot \log_{10}\left(\frac{\max(\phi_{original})}{\text{RMSE}}\right). \tag{2.15}$$

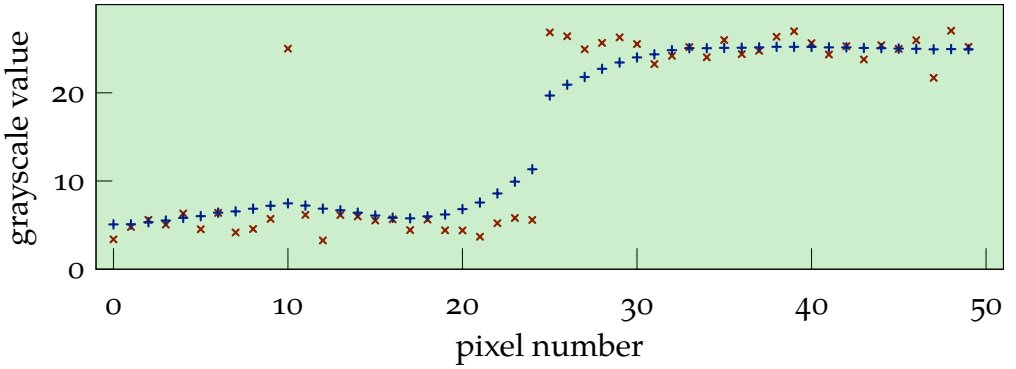
All filter parameters listed above were chosen to minimize RMSE and maximize PSNR. Results are shown in Tab. 2.1 and will be discussed in the next section. Image examples are provided in Fig. 2.12.



(a) Threshold of $\tau = 10$.

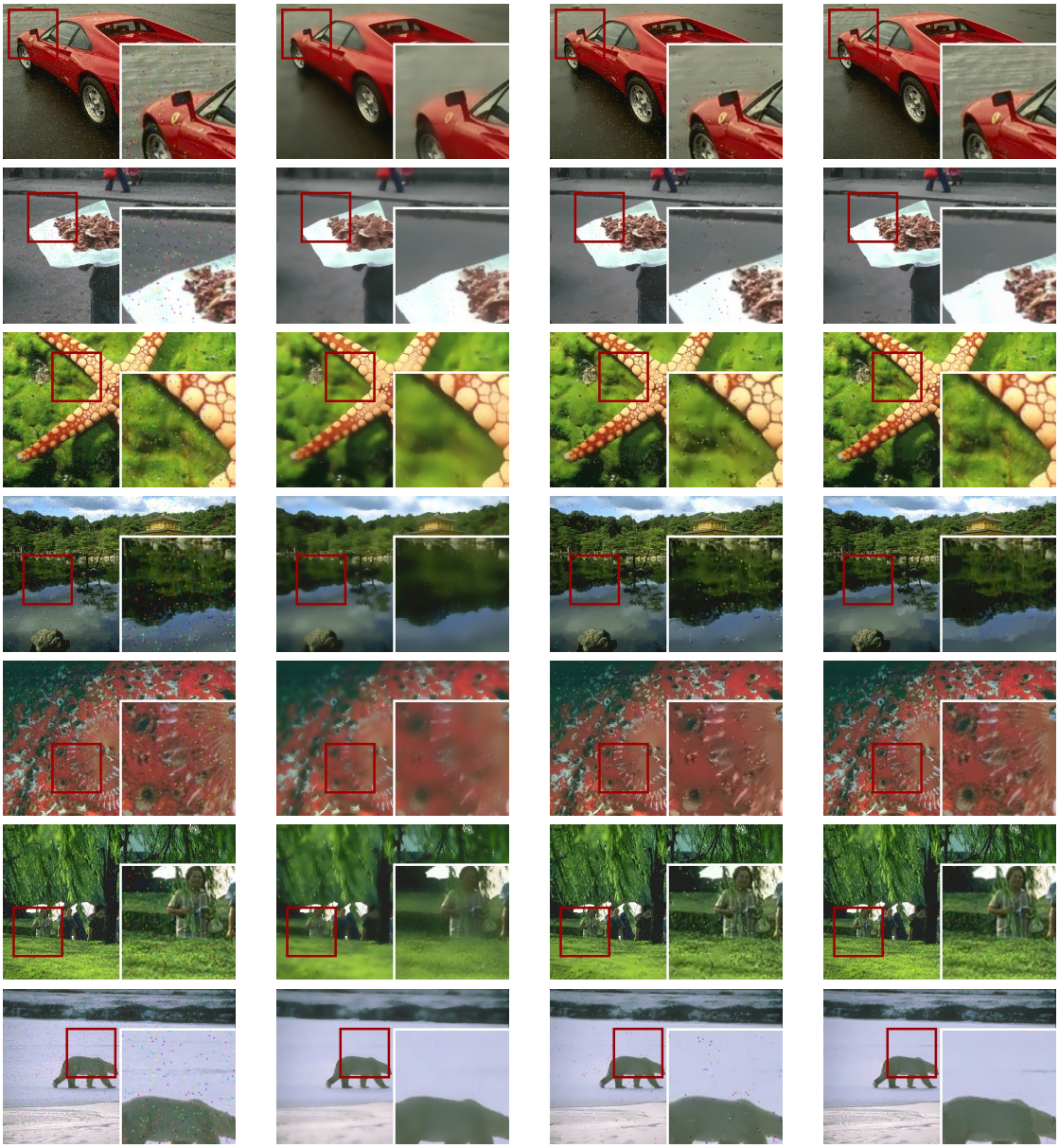


(b) Threshold of $\tau = 15$.



(c) Threshold of $\tau = 30$.

Figure 2.11.: Shown is the effect of three different thresholds on one data set: A one dimensional grayscale image containing a color edge at pixel 25 and one outlier at pixel 10. Detailed explanations are shown in text, see Sec. 2.3.2.



(a) Noisy image. **(b)** Bilateral Filter. **(c)** NLM Filter. **(d)** Proposed EPF.

Figure 2.12.: Visual comparison of filter results. Quantitative results are shown in Tab. 2.1. Images taken from Berkeley Image Data Set [10].

The proposed filter achieves on both data sets the best performance markers. In the discussion, see Sec. 2.4, the results are also compared to more recent, state-

	Berkeley Data Set		Coco Data Set	
	RMSE	PSNR	RMSE	PSNR
Original	17.95	23.31	17.31	23.33
EPF	**7.06**	**31.05**	**7.89**	**30.47**
Bilateral	10.41	27.75	10.43	28.01
Gaussian	14.59	25.08	15.69	24.81
Median	14.04	25.63	14.91	25.54
NLM	11.40	26.86	12.28	26.44

Table 2.1.: RMSE and PSNR computed on the Berkeley Data Set (500 images) and the Coco Data Set (40775 images). The first line "Original" refers to the not denoised image. The error is ± 0.01 for all values.

of-the-art algorithms.

2.3.4. Denoising of 1d sensor data

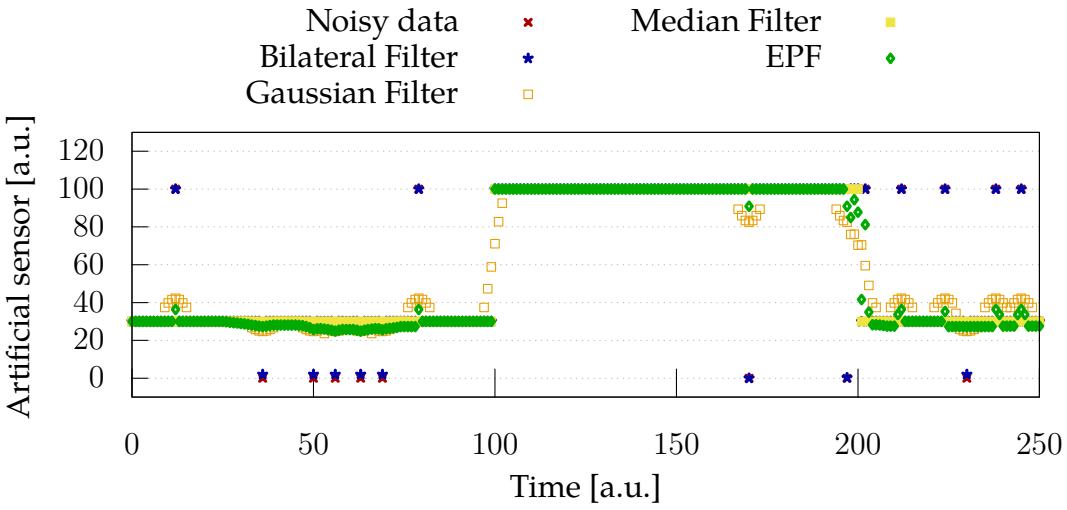
As already suggested in Fig. 2.6, Fig. 2.10, and Fig. 2.11, the filter can also be applied to 1d data. This may happen, for example, as a post-processing step for sensor readings. The filter is tested on three different settings: first, an alternating line, which switches every 100 samples its height to either $f(x) = f_{min}$ or $f(x) = f_{max}$; second, a sawtooth wave defined by $f(x) = x - \text{floor}_{100}(x)$; and third, a sinusoidal wave $f(x) = \sin 2\pi x/250$ with a wave length of 250 data points. Every data line consists of a total length of 1000 samples. To each scenario either Gaussian noise with variance of $\sigma = 10$, salt-and-pepper noise (to 5% of samples), or both is added. Visual examples are shown in Fig. 2.13, Fig. 2.14, and Fig. 2.15.

Again, the proposed filter ($N = 11$, $\tau = 30$) is compared to a Gaussian blurring filter (kernel: 7×7 px, $\sigma_{x,y} = 3$), a bilateral filter ($\sigma_c = 30$, $\sigma_s = 30$, and a median filter (kernel size: 9 px). RMSE and PSNR is computed according to Eqn. (2.14) and Eqn. (2.15). On each setting 1000 trials are performed and averaged. Results are shown in Tab. 2.2.

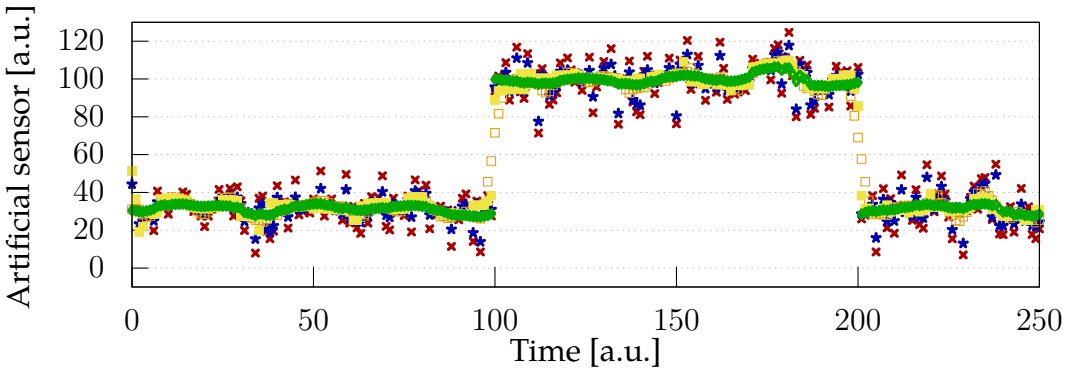
		Gauss		s&p		Gauss and s&p	
		RMSE	PSNR	RMSE	PSNR	RMSE	PSNR
1) Alternating Line	No denoising	10.0	22.3	15.2	16.4	18.1	17.2
	EPF	**2.6**	**32.3**	**3.9**	**28.5**	**5.1**	**26.7**
	Bilateral	6.9	25.3	15.2	16.4	16.5	17.7
	Gaussian	6.0	25.3	7.8	22.2	8.7	22.1
	Median	5.1	26.8	4.1	27.9	6.0	25.4
2) Sawtooth Wave	No denoising	10.0	21.6	14.1	17.1	17.1	17.0
	EPF	**3.7**	**29.3**	**4.6**	**26.5**	**6.6**	**24.5**
	Bilateral	7.0	24.4	14.0	17.1	15.5	17.5
	Gaussian	7.6	22.6	8.8	20.9	9.6	20.6
	Median	5.7	25.2	5.8	24.9	7.4	23.1
3) Sinusoidal Wave	No denoising	10.0	20.1	12.9	17.8	16.1	16.0
	EPF	**2.7**	**29.5**	2.6	29.8	**4.4**	**25.6**
	Bilateral	6.7	23.0	12.8	17.9	14.4	16.9
	Gaussian	3.9	26.7	5.0	24.2	6.3	22.6
	Median	4.1	26.1	**0.6**	**46.1**	4.5	25.6

Table 2.2.: Comparison of RMSE and PSNR computed on three different scenarios: 1) an alternating line, 2) a sawtooth wave, and 3) a sinusoidal wave. To each scene three different noise types (Gaussian, salt-and-pepper (s&p), or both) are added, resulting in 9 different experiments. Each experiment is repeated 1000 times and averaged; the error is ± 0.1 for all values.

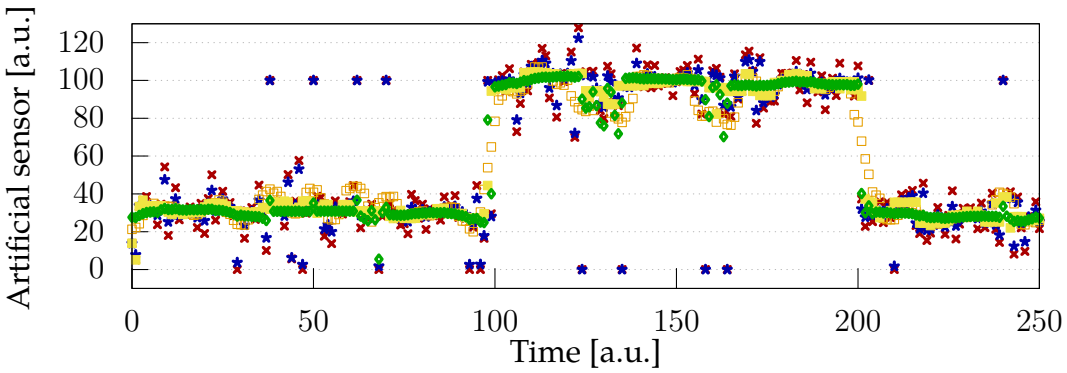
In almost all experiments EPF outperforms other standard 1d filtering methods. While the median filter performs very well on salt-and-pepper noise, it is not edge preserving and thus introduces artefacts on edges. The bilateral filter on the other hand, handles edges very well, but has significant trouble with removing salt-and-pepper noise. The proposed EPF filter performs well on both, Gaussian and salt-and-pepper noise and is edge preserving.



(a) Only salt&pepper noise, density $p = 5\%$.

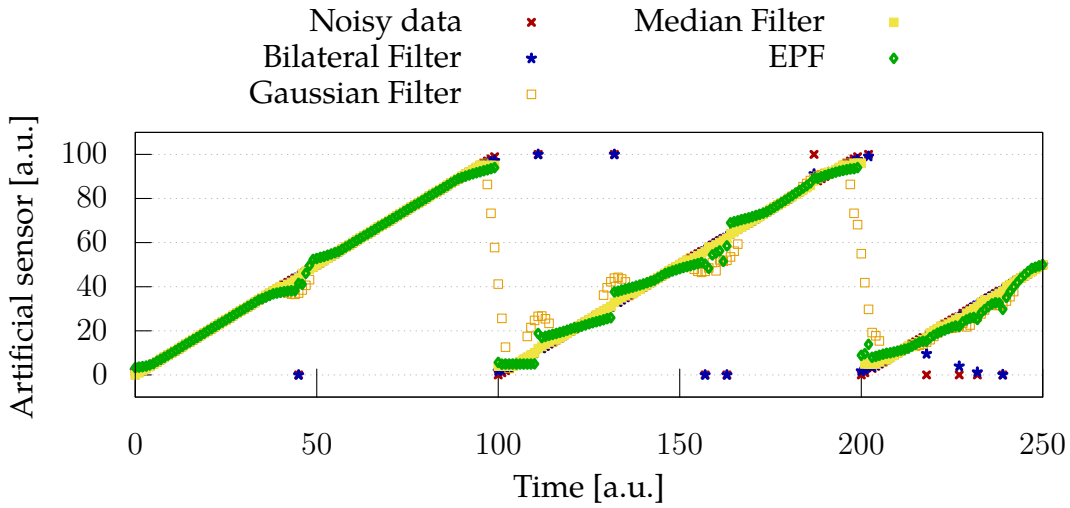


(b) Only Gaussian noise, mean of 1 and standard deviation $\sigma = 10$.

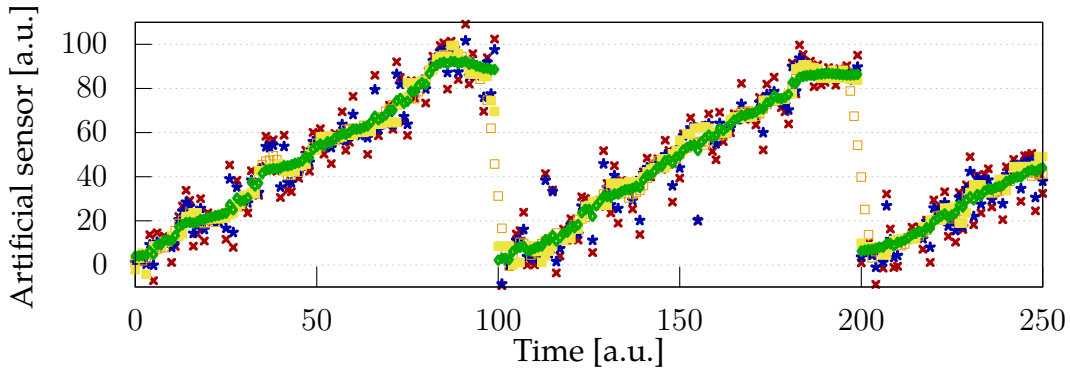


(c) Both, salt&pepper (density $p = 5\%$) and Gaussian noise (mean of 1 and $\sigma = 10$).

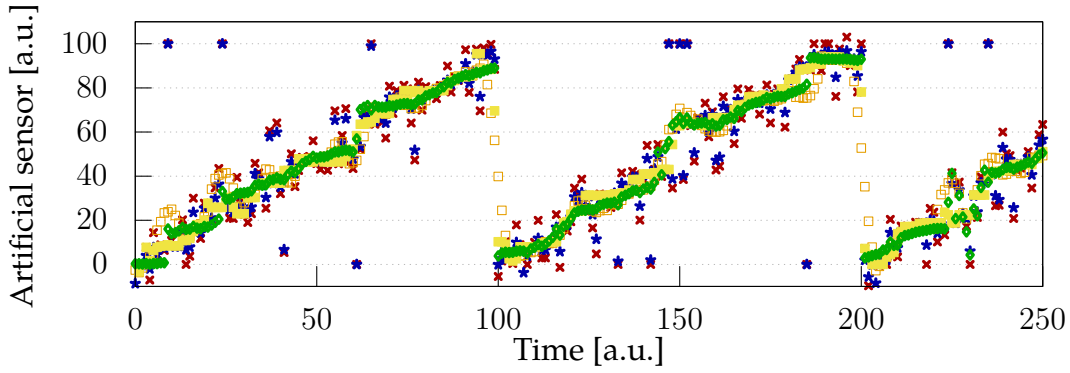
Figure 2.13.: Examples of different denoising algorithms on stepwise data.



(a) Only salt&pepper noise, density $p = 5\%$.

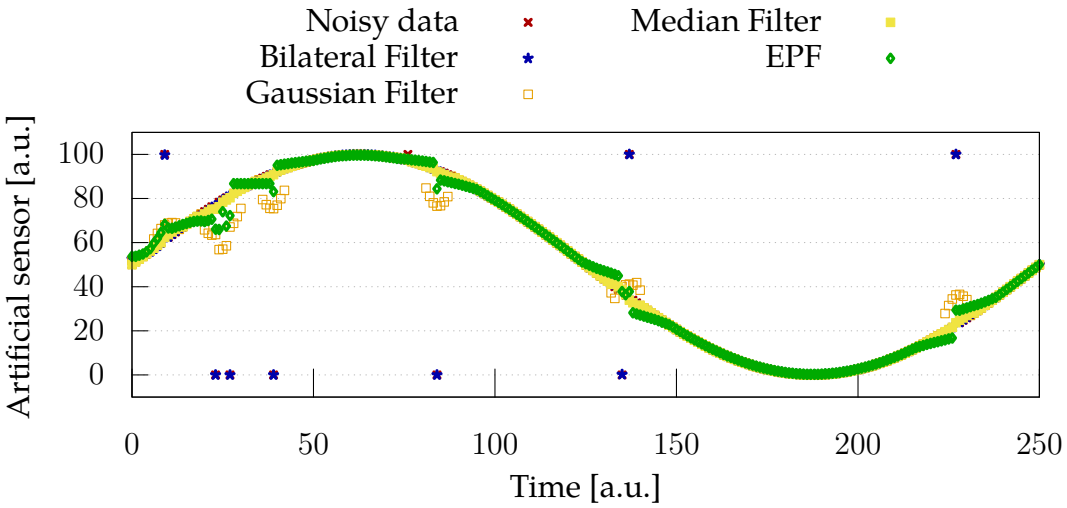


(b) Only Gaussian noise, mean of 1 and standard deviation $\sigma = 10$.

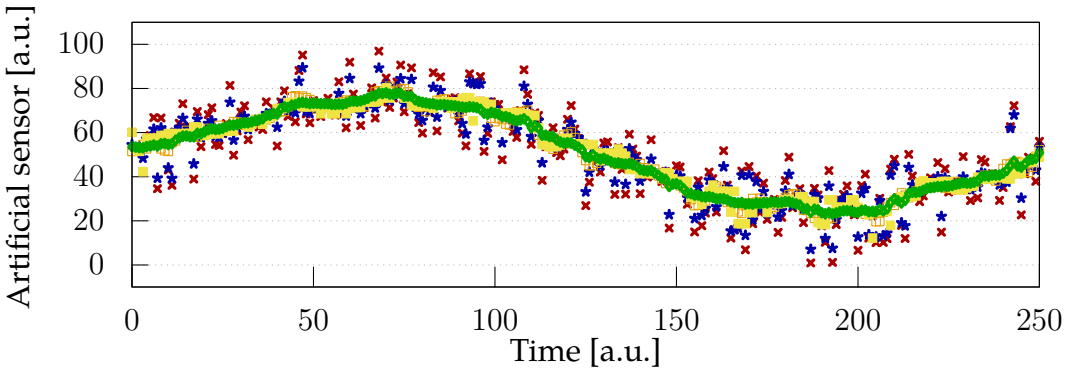


(c) Both, salt&pepper (density $p = 5\%$) and Gaussian noise (mean of 1 and $\sigma = 10$).

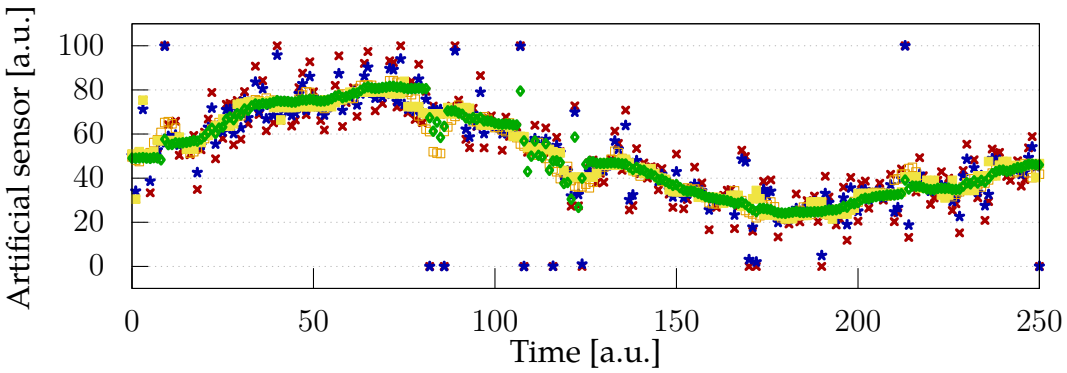
Figure 2.14.: Examples of different denoising algorithms on sawtooth data.



(a) Only salt&pepper noise, density $p = 5\%$.



(b) Only Gaussian noise, mean of 1 and standard deviation $\sigma = 10$.



(c) Both, salt&pepper (density $p = 5\%$) and Gaussian noise (mean of 1 and $\sigma = 10$).

Figure 2.15.: Examples of different denoising algorithms on sinusoidal data.

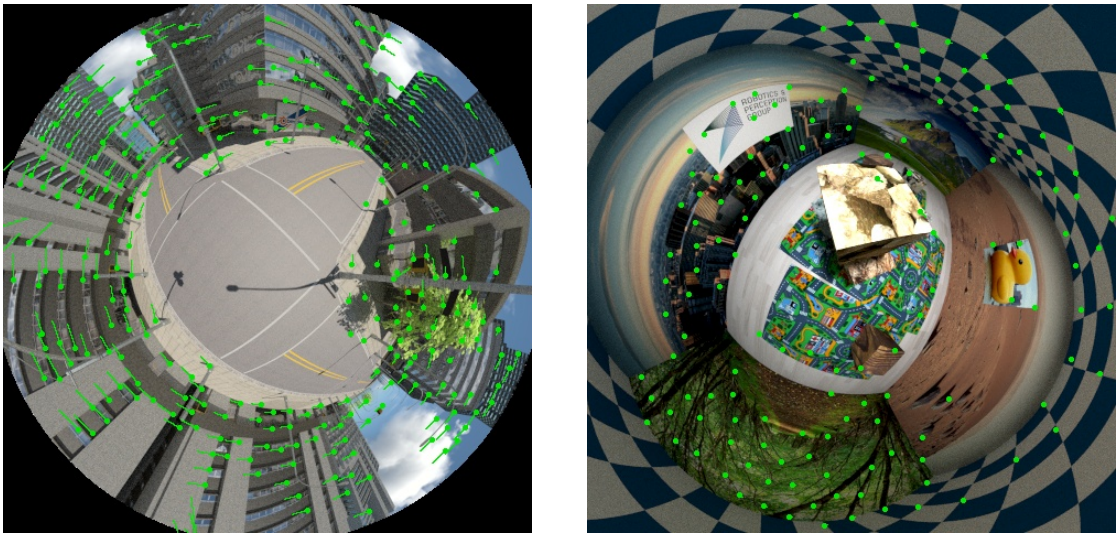
Image Size [px]	EPF CPU [Hz]	EPF GPU [Hz]	BM3D CPU [Hz]
240×180	2.0	80.4	
320×240	1.1	48.0	
480×320	0.5	23.8	0.4
640×480	0.3	12.4	
800×600	0.2	7.7	0.1
1024×768	0.1	4.2	0.1

Table 2.3.: Time performance for images of different sizes. The test images were taken from the validation set of the Berkeley Segmentation Data Set and Benchmark [10]. 100 measurements were taken and averaged. The proposed EPF filter is compared to state-of-the-art algorithm BM3D [27] as shown in [111]. BM3D is, according to [111], one of the fastest recent methods. The error is ± 0.1 for all values.

2.3.5. Time performance of denoising algorithm

Average frame rates for differently sized images are computed in Tab. 2.3. 100 images from the validation data set from [10] were used and the results averaged. As shown in Sec. 2.2.2 the computational complexity does not depend on the threshold and rather increases linearly with frame and subwindow size. In this test, a subwindow size of 10×10 px is used. Results are shown in Tab. 2.3.

Two implementations of the algorithm are tested: The CPU measurement refers to a single-threaded implementation using an Intel i7-3930K twelve-core processor at 3,2 GHz using one core and 16 GB RAM. The GPU version is executed on an Nvidia GTX580 graphics card using 512 cores and 1.5 GB device memory. The GPU implementation for all frame sizes is about 40 times faster than the CPU implementation. However, the CPU implementation is rather naive and still open for improvements. For images of size 480×320 px real-time performance is achieved.



(a) Urban canyon scenario.

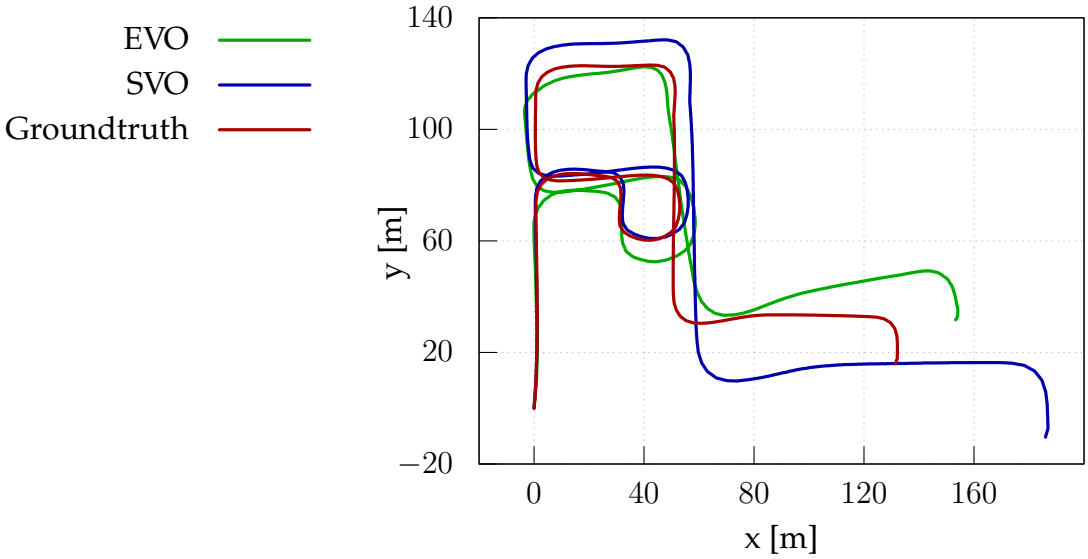
(b) Indoor scenario.

Figure 2.16.: Urban canyon and indoor scenario with sparse optical flow (visualized as green dots and lines).

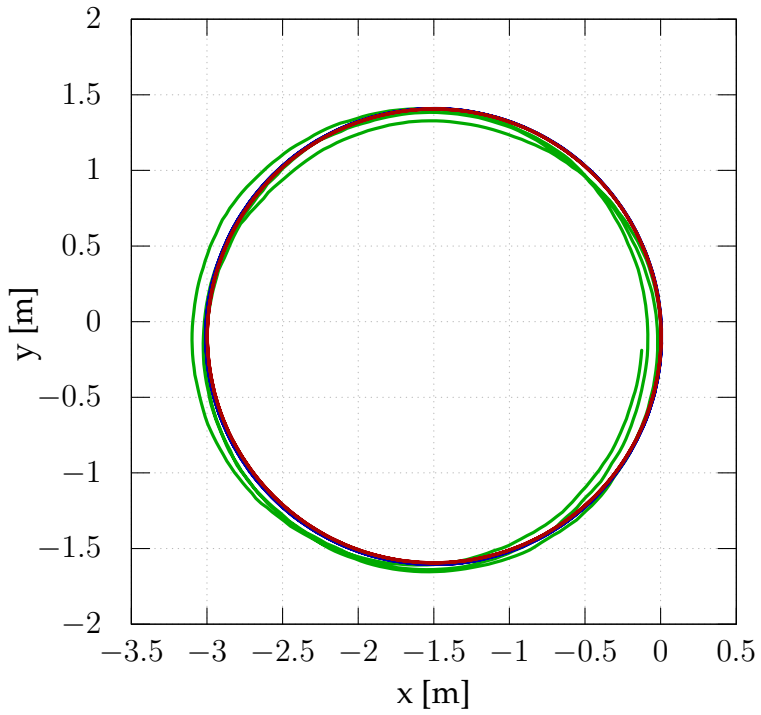
2.3.6. Visual Odometry in simulation

Ground truth generation for fast flying AAVs still poses a big problem. External tracking systems are confined to single rooms and are expensive. Furthermore, it is hard to distinguish, which problems arise from the visual odometry algorithm and which stem from the flight controller itself.

EVO is first benchmarked on two simulations: an "Urban Canyon" and an "Indoor" scene. The "Urban Canyon" contains a 400 m long flight through an artificial city, while the "Indoor" scene is a circular path that exactly repeats thrice (details in [137]). Example frames from both scenes are shown in Fig. 2.16. A visual comparison of the "Urban Canyon" and "Indoor" trajectory can be found in Fig. 2.17.



(a) Trajectory of the "Urban Canyon" scenario in the x-y-plane.



(b) Trajectory of the "Indoor" scenario in the x-y-plane.

Figure 2.17.: Overview of the simulation results as computed by the EVO algorithm proposed here. It is compared to state-of-the-art SVO algorithm [44].

	"Urban Canyon" 400 m length		"Indoor" 28 m length	
	RMSE [m]	Displacement [m]	RMSE [m]	Displacement [m]
EVO	$\mathbf{10.66 \pm 0.01}$	$\mathbf{27.12 \pm 0.01}$	0.91 ± 0.01	0.14 ± 0.01
SVO	19.13 ± 0.01	60.61 ± 0.01	$\mathbf{0.14 \pm 0.01}$	$\mathbf{0.004 \pm 0.01}$

Table 2.4.: Results of the two simulation scenes "Urban Canyon" and "Indoor" [137]. Shown is RMSE, which measures the total difference of the entire flight trajectory compared to the ground truth information. Displacement holds the euclidean distance between ground truth finish position and estimated finish position.

Root-Mean-Square Error is used to compute the translation error as defined:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n} (\vec{\hat{s}}_t - \vec{s}_t)^2}{n}}. \tag{2.16}$$

RMSE measures the total difference throughout the entire flight trajectory. Additionally, the final displacement between the ground truth finish position and the algorithm's finish position is computed. This is a measure for the cumulative error. Results can be found in Tab. 2.4. A visual representation of the translation errors is shown in Fig. 2.18 and Fig. 2.19.

The proposed method outperforms the current state-of-the-art SVO algorithm by about 30 m in the more demanding urban canyon data set. On the other hand, the much smaller and more repetitive indoor scene, a circle of 3 m diameter, which is flown for three times, can be solved better by the SVO algorithm. One possible reason is, that SVO does not compute the pose update on all frames. In SVO, a pose update is only computed on selected key frames. Between two key frames the pose is updated based on the latest key frame. In a scenario without any turns, as it is the case for a circle, this will lead to a close-to-perfect solution. In any real world scenario this reduced pose update is prone to failure, as can be seen in the urban canyon scenario, see Fig. 2.18.

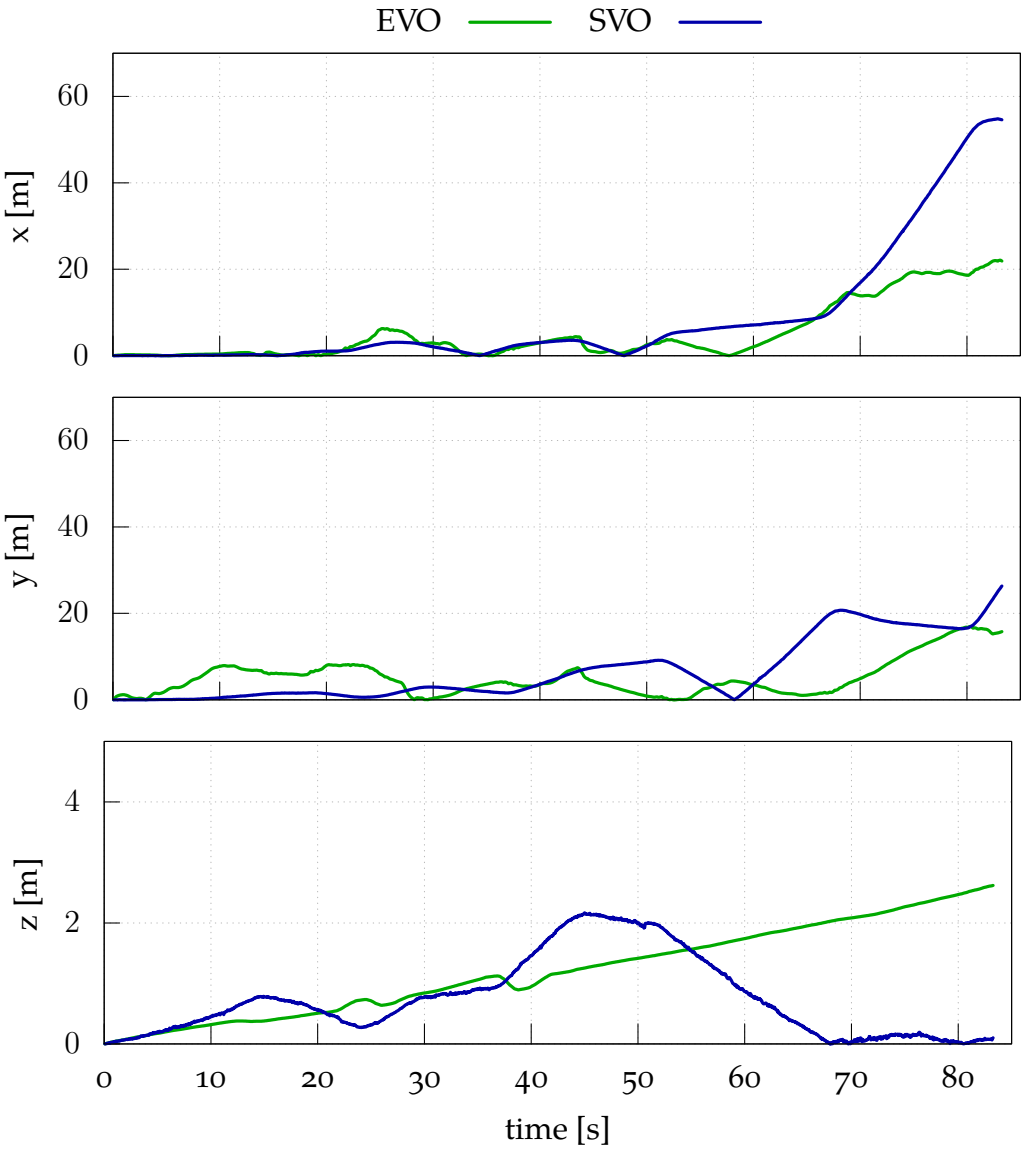


Figure 2.18.: Translation error x, y, and z of the "Urban Canyon" trajectory shown in Fig. 2.17a.

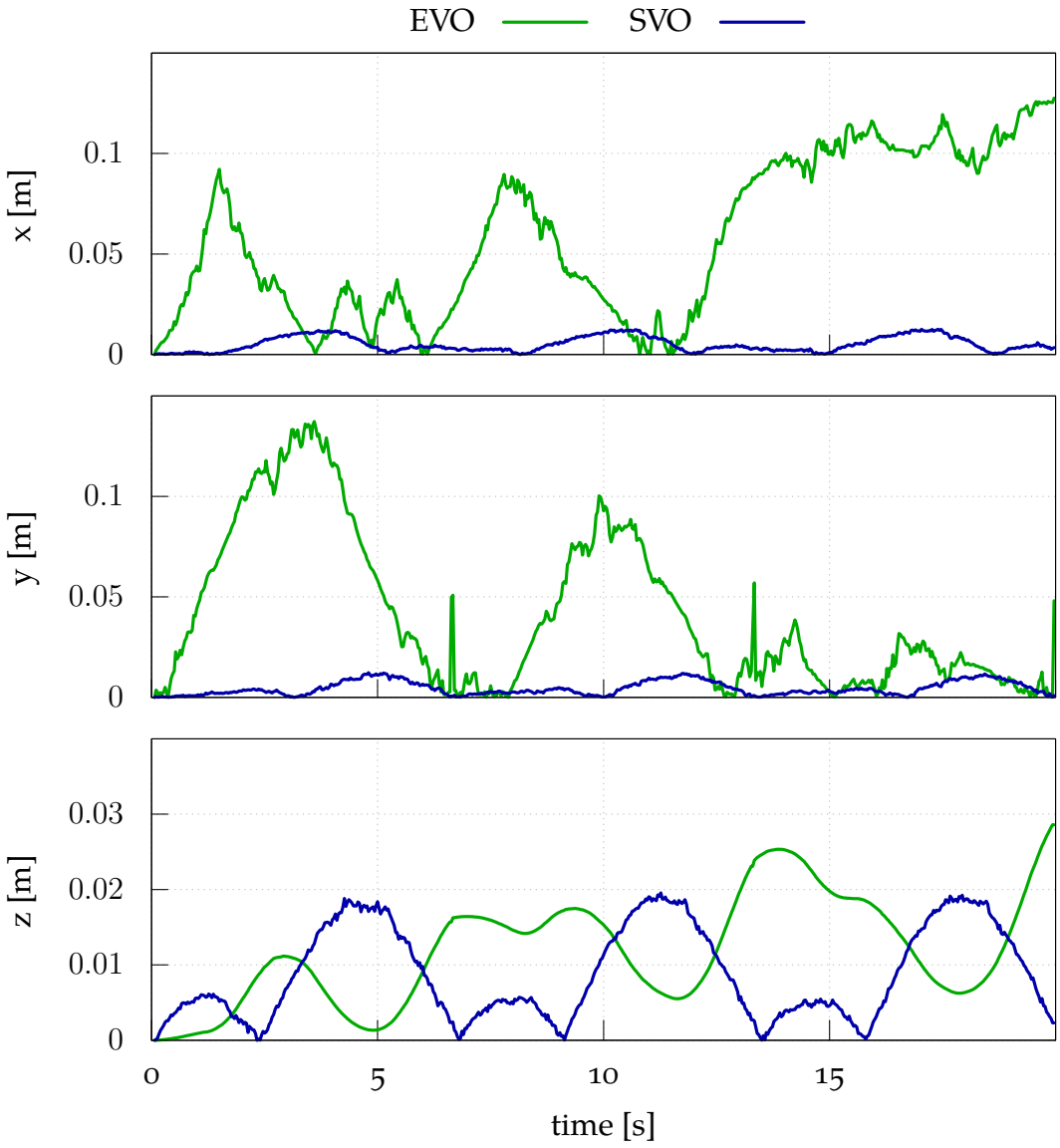


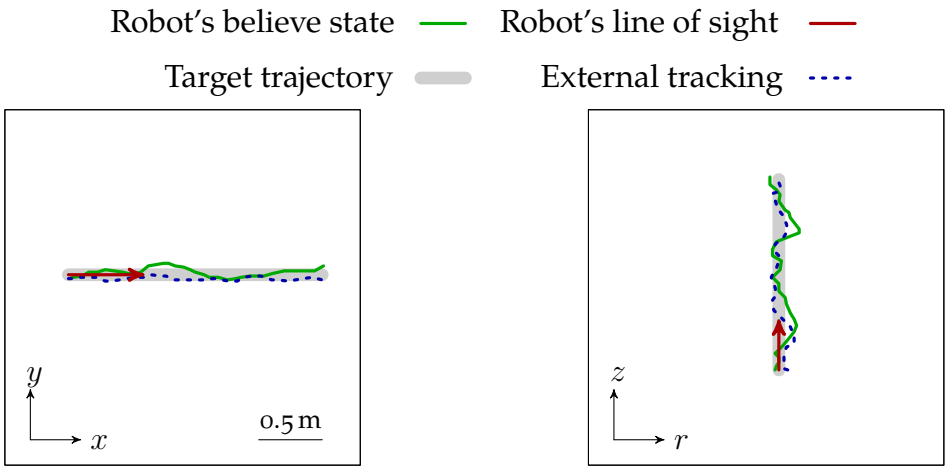
Figure 2.19.: Translation error x, y, and z of the "Indoor" trajectory shown in Fig. 2.17b.

On the other hand, EVO contains a systematic error estimating the robots height as can be seen in Fig. 2.18. The drone continuously lifts off for the entire track. This might be due to the simulation environment. The images contained in the benchmark are simulated using a catadioptric camera as shown in Fig. 2.16. The benchmark setup is not perfectly equivalent to the system assumed in the approach and as a consequence some far-away features are estimated too low.

2.3.7. Externally tracked indoor flights

Now that EVO's performance is evaluated on a benchmark, its pose information is fused with IMU data via a Kalman filter. First, the robot is moved manually to eliminate problems stemming from the flight control algorithm. These problems include rapid movements containing large feature offsets, which also poses problems for the IMU, and sharing computing power with the flight controller. Then, the same trajectories are flown in full flight mode. Six scenarios are devised:

1. a straight line in the $x - y$ plane with length 2 m,
2. a straight line upwards into the z-direction with length 1.5 m, i.e. lift off,
3. a square with side length 2 m, the UAV always pointing into the direction of flight,
4. a square with side length 2 m, the UAV always pointing into the same direction,
5. a circle with diameter 2 m, the UAV always pointing into the direction of flight, and
6. a circle with diameter 2 m, the UAV always pointing into the same direction.



(a) Line in $x - y$ plane. The plotted scale holds for all figures.

(b) Lift off in z-direction plotted against the radius r.

(c) Square, quadrocopter pointing into direction of flight.

(d) Square, quadrocopter always pointing into the same direction.

(e) Circle, quadrocopter pointing into direction of flight.

(f) Circle, quadrocopter always pointing into the same direction.

Figure 2.20.: Qualitative examples of recorded target trajectory.

Scenario	Manual mode [m]	Flight mode [m]
1)	0.03 ± 0.01	0.07 ± 0.03
2)	0.06 ± 0.03	0.06 ± 0.03
3)	0.07 ± 0.04	0.08 ± 0.04
4)	0.05 ± 0.02	0.09 ± 0.04
5)	0.06 ± 0.03	0.11 ± 0.05
6)	0.04 ± 0.02	0.10 ± 0.04
Average	0.05 ± 0.03	0.09 ± 0.04

Table 2.5.: For each of the six trajectories (which are shown in Fig. 2.20) ten trials were performed and the averaged RMSE in the x-y-plane for these trials is shown. In "manual mode" the quadrocopter was moved manually on the trajectories to eliminate problems from flight control algorithms. In "flight mode" trials were performed in full flight mode.

For each scenario ten trials were recorded, resulting in total in 120 runs. An example of each trajectory is shown in Fig. 2.20. To achieve a meaningful evaluation, ground truth information was generated utilizing a 3d Asus Xtion Pro camera for external tracking. The camera only offers reasonable data in ranges smaller than 3.5 m [53]. Thus, only short scenarios were used. All results are shown in Tab. 2.5.

Still, this type of evaluation remains problematic. As shown in [31, 53] the Asus Xtion Pro's depth resolution is 640×480 px. This already leads to a theoretical limit of 14 mm of separation distance at a camera distance of 2.5 m. However, sensor noise worsens the measurement significantly [31, 53]. The errors shown in Tab. 2.5 are accumulated based on IMU and VO only. Ground truth is considered free of error, even though this is clearly not true.

One could do a maximum error estimate of the camera recordings:

- Voxel position in the field of view: The Asus Xtion Pro creates a pattern of structured infrared light. The reflection is recognized by an infrared sensor, which computes the voxel's depth. The pattern is much more accurate in front of the camera than on the border of the 57° field of view. Registration,

the fusing of depth and RGB data, is much more accurate in the center of the pattern.

- Temperature significantly increases the cameras error rate. This type of noise is uniform and does not depend on the voxel position [53].

- Lastly, illumination from infrared sources, e.g. the sun and many light bulbs, disrupts the structured light pattern. Voxels, which are not recognized on the pattern are extrapolated from neighboring voxels. No information is presented to the user whether the raw sensor data contains a real reading or an extrapolation to the user. Thus, quantifying an error here is almost impossible.

Heuristically, an error of up to ± 1 cm was found in front of the camera and an error of up to ± 2.5 cm was found at the edge of the horizontal field of view in an office environment. To solve the camera problem, one could use a predefined trajectory. This, however, only works for the manual mode and not for full flight mode. There are numerous commercial motion capturing solutions available. These system are either based on external tracking from multiple RGB cameras (the most prominent system is called Vicon [131]), or based on sensors placed directly on the robot fusing IMU, VO, and GPS (e.g. [124]). However, the most prominent feature of the Asus camera is its comparatively low cost of about 120 US$ while inexpensive commercial solutions may cost hundred times as much.

2.3.8. Office indoor flight

Next, a real world indoor office flight is shown. The trajectory can be found in Fig. 2.21. The flight's duration is 38.4 s. The goal was to fly a figure-of-eight around a central obstacle and avoid all obstacles and walls, which resulted in the here-shown difficult trajectory. The real world flight adds additional challenges as errors from the flight controller are introduced.

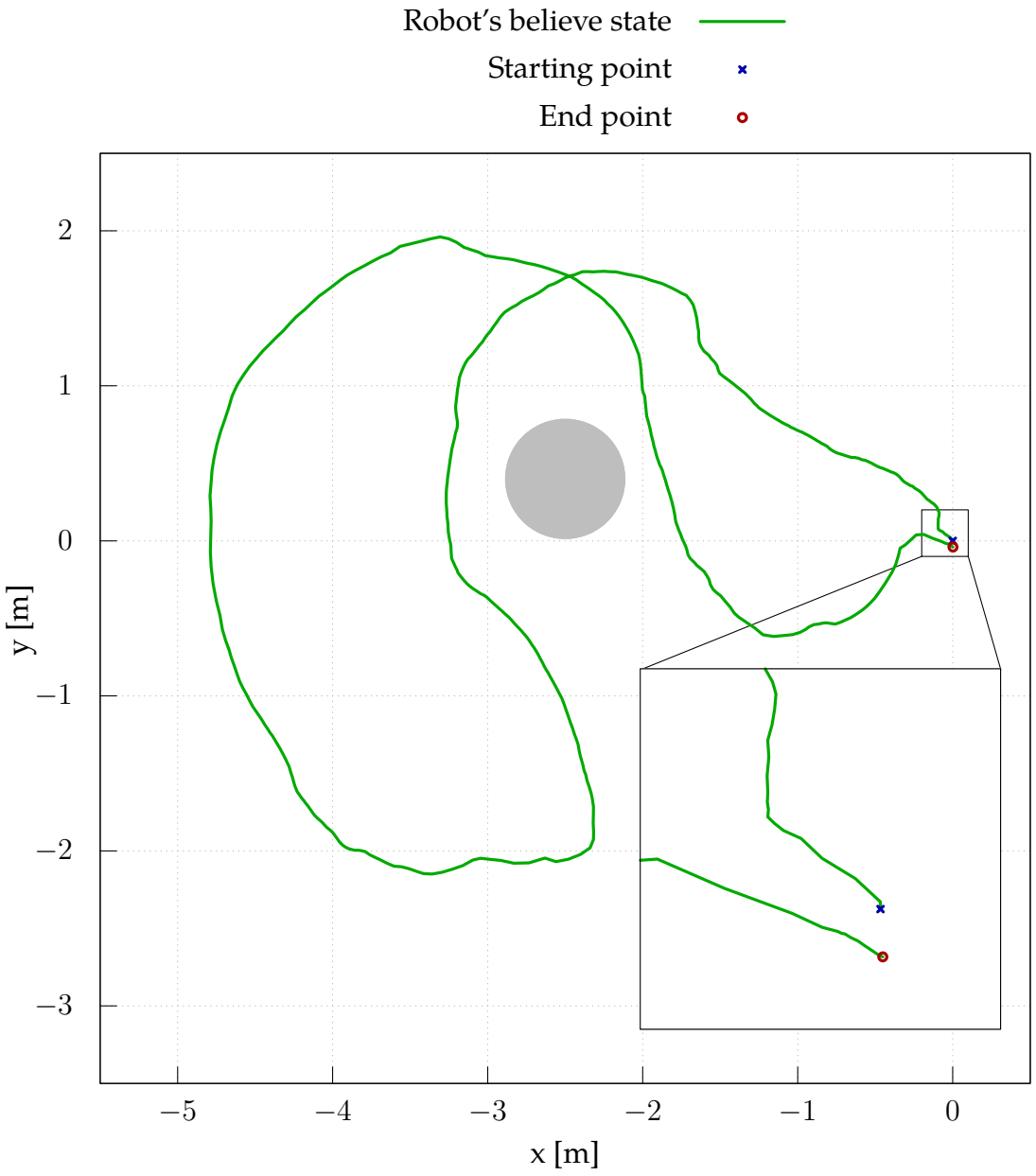


Figure 2.21.: The robot started and landed at position $(0, 0)^T$ and flew a figure-of-eight around a central obstacle shown in gray. The trajectory (in green) shows the internal believe state of the robot (fusion of VO and IMU); it is 19.4 ± 0.1 m long. It took the robot 38.4 s to fly the track. The starting point is marked with a blue cross, the estimated landing position with a small red circle, while the real landing position was again at $(0, 0)^T$.

		Intel-i7 [ms]	ARM Cortex-A53 [ms]
EVO	Feature extraction	2.5 ± 1.9	14.3 ± 3.2
	Optical flow	1.9 ± 1.2	10.6 ± 2.2
	Motion computation	7.8 ± 1.5	33.6 ± 7.3
	Depth filter update	2.4 ± 0.9	12.4 ± 2.8
	Total	14.6 ± 5.5	70.9 ± 15.5
		Intel-i7 [ms]	ARM Cortex-A9 [ms]
SVO	Total	3.04 ± 1.10	18.17 ± 6.0

Table 2.6.: Average time consumption in milliseconds by individual components of the algorithm on the data set. Comparison between run times on a laptop (Intel Core i7 (2.80 GHz) processor and the Raspberry Pi (ARM Cortex-A53). It is compared to the SVO algorithms results as shown in [44].

While the robot started and landed on the same spot, the internal believe state of the robot shows a small displacement. Please note, that for this experiment the vision odometry, as well as other onboard sensors of the robot were used; namely a gyroscope and accelerometer. They are fused with the vision algorithm via an EKF as shown in Fig. 2.7. Thus, one can compute the euclidean distance between the start and finish position as a measure for accuracy. In the $x - y - z$ plane the final displacement is 0.10 ± 0.01 m, which results in a relative error of only $0.5 \pm 0.1\%$. The offset in the $x - y$ plane is computed to 0.04 ± 0.01 m.

2.3.9. Time performance of Visual Odometry algorithm

Results for frame rates are shown in Tab. 2.6. The novel approach presented here is compared to the SVO [44] algorithm.

On a first glance it seems, that EVO provides a significantly worse time performance than SVO. Please note however, that SVO does not compute a full position update for every frame, but rather on selected key frames. This leads to a trade-off between fast reaction times and pose accuracy on the one hand, and computational power, which can be used for other processes, on the other hand. Additionally, SVO uses a more powerful processor. While the chosen approach

— full pose update on every frame — is more conservative, it leads to better results in the long run. The question of how many frames need to be analyzed for stable results is part of ongoing research in our lab.

2.4. Discussion

2.4.1. Edge-Preserving Filter

Here, a novel real-time edge preserving denoising filter named EPF is presented, which replaces noisy areas by uniformly colored patches. Performance is significantly better than other standard methods on 2d images. Artificial 1d data shows similar results.

In [52] a comparison to other methods is given, including state-of-the-art methods like BM3D [27], EPLL [139], or LSSC [74] based on the Berkeley Data Set [10]. All these methods exploit the image nonlocal redundancies, in contrast to EPF, which uses a local neighborhood. In Tab. 2.7 a comparison between the proposed EPF algorithm and other state-of-the-art methods is shown. Clearly, the proposed method performs slightly worse than other recent algorithms. On the contrary the review [111] performs a conclusive study on computational complexity. According to this work, one of the fastest algorithms, BM3D, manages to denoise not more than one image sized 256×256 px per second. This is far from real-time and not feasible for robotic applications or critical sensor readings. A comparison to the proposed EPF filter is shown in Tab. 2.3.

This means, the proposed EPF performs only slightly worse than recent denoising methods, but offers real-time performance, which makes the filter applicable to video streams and hence can be used in the future as a component inside the perception-action loop of robotic applications. It enables image processing and data filtering on embedded hardware, for example in flying robots, which is another research area of ours. The filter not only works well in the image domain,

Gaussian Noise	Recent Methods	EPF
$\sigma = 10$	$33.5 - 34.8$	30.7
$\sigma = 30$	$27.8 - 29.2$	23.0
$\sigma = 50$	$25.1 - 26.8$	19.9
$\sigma = 100$	$21.6 - 23.6$	15.5

Table 2.7.: PSNR values computed on the Berkeley data set for state-of-the-art methods (as shown in [52]) compared to the proposed EPF filter.

but can be extended to data of any dimension, e.g. noisy 6d point cloud data. This is demonstrated in this work by filtering 1d sensor data.

2.4.2. Embedded Visual Odometry

A novel lightweight omnidirectional camera setup for fast moving robots is investigated. All computations are performed online on the robot. VO is evaluated using two simulated environments. Then, VO is combined with IMU data. The resulting pose information is confirmed on a real robot using external tracking and via measuring the final displacement during an office flight.

The achieved frame rate of 15 ± 3 Hz, as shown in Tab. 2.6, is sufficient for real-time applications in autonomous agents with low-power hardware. A real world example gives an understanding of the error margins: A self localization error of 0.1 m and a frame rate of 15 Hz is assumed. Furthermore, we will assume that the AAV needs 5 frames to detect an obstacle and initiate counter measures. Using the given frame rate of 15 Hz, the quadrocopter needs approximately 0.33 s to detect an obstacle. Within these 0.33 s the safety error margin of 0.1 m (the above localization error) must not be met. Thus, we can survey in indoor environments with a maximum velocity of approximately 1.1 km/h. This allows for a broad range of applications, e.g. fast search and rescue in impassable terrain.

First, the EVO algorithm is tested in simulation on two benchmarks. It is shown that it performs significantly better than current state-of-the-art methods.

Next, the robot is tracked via an external camera system. The deviation between external tracking and the robot's internal state of believe was found to be 5 ± 3 cm in manual mode on average; in real flight self localization performs at 9 ± 4 cm. The utilized external tracking system performs already with an error of at least ± 1 cm [53] and thus introduces significant uncertainty.

As in VO algorithms the error accumulates over time and is hard to measure on short trajectories; therefore, a real world office flight with a trajectory of 19.4 ± 0.1 m length is flown. While the robot started and landed on the same spot, the robot's internal believe state shows a displacement of about 0.10 ± 0.01 m. This leads to an error of $0.5 \pm 0.1\%$, which is about the same as in the "Indoor" simulation environment. Here, the robot had a displacement of 0.14 ± 0.01 m on a 28 m trajectory. For comparison: the 400 m "Urban Canyon" data set contains an error of approximately 6.8%.

This work enables autonomous robots to localize themselves, while allowing at the same time to build a depth map. This map offers for example obstacle avoidance or mapping capabilities. All computations are performed online on embedded hardware, meaning that the robot is able to work in unknown environments. It can support autonomously, for example, in search and rescue mission, disaster relief work, or exploration tasks.

This concludes the first part of this thesis. In the second part, the focus lies on the action side of the action-perception loop. It is shown that using a low level, bottom-up method, sophisticated high level planning is feasible.

3

Action planning in robots

3.1. Introduction

In the last chapter it was shown that a bottom-up method is already powerful enough for complex perception tasks. A fast moving robot can, without external tracking or external computations, perform safe indoor flights. However, the robot from the last chapter has no knowledge about higher level symbols like "door", "window", or "human being". It treats all objects as obstacles alike and performs avoidance maneuvers around them.

The step from symbols to the robot's state space, which is made up of low level motor signals and sensor readings, is called signal-to-symbol gap. In the next chapter a method is introduced, which will bridge this gap based on a bottom-up method called Semantic Event Chains (SECs) [3, 7]. It generalizes well and it is shown that it allows for complex planning. The action state space of the flying drone is very limited. Thus, this chapter deals with a two armed robot: two Kuka Lightweight Robot arms [18] mounted on a table as shown in Fig. 3.1. There is one three fingered adaptive robot gripper made by the Schunk company [60] attached to each arm. In this work, only one arm is used since bimanual manipulations are not considered. As additional sensor input two Asus Xtion Pro cameras [53], one pointing at each arm, and one Nikon D7000 high resolution RGB Digital Single-Lens Reflex Camera (DSLR) are used.

In recent years, developments in the field of robotics have emerged solutions to a very wide field of problems: Ranging from autonomous driving to search

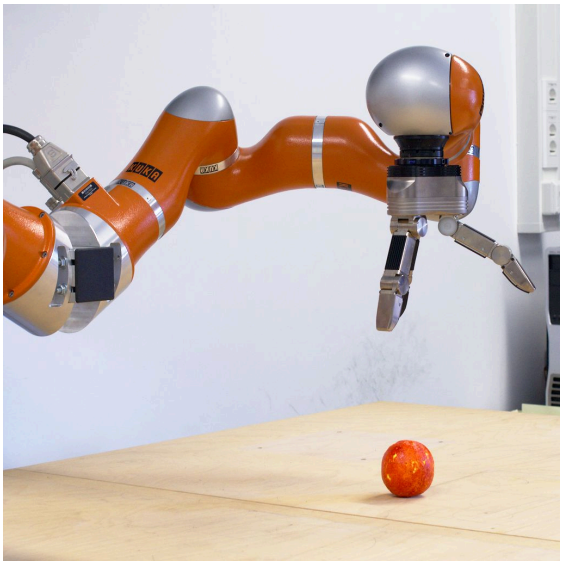


Figure 3.1.: One of the two Kuka Lightweight Robots [18]. Connected to the robot arm is a three-fingered gripper.

and rescue missions in remote areas; from space exploration tasks to the care of elderly people. This also means that robots become more and more integrated into our daily life. Along come increasingly complex situations, which need to be mastered. Planning plays a vital part here.

Historically, planning in robotics is divided into two fields: on the one side there are voltages and currents to be controlled. In this domain physical constraints are most important, i.e. robot hardware, collision detection, collision avoidance, and a dynamically feasible trajectory [25, 93]. On the other side there are higher level descriptors, which work in the symbolic domain. One example for two symbols might be *empty cup* and *cup full of water*. Now, a transfer function *fill cup with water* to arrive at the second symbol from the first can be defined. Furthermore, the fact that the cup must be empty in order to fill it with water is called precondition. Preconditions ensure that only transfer functions are applied to symbols, where it makes sense. For example, filling a full cup with more water would result in spilling. Similarly, the filled cup is called postcondition of the filling action [50, p. 66f., p. 335f.]. The transition from raw sensor data to a symbolic descriptor is again the above mentioned signal-to-symbol gap and both

planning approaches are separated by this gap. They are considered as two different problems [11, 17, 89], where the first one is bottom-up and the second one top-down. In this work, the gap is bridged using a bottom-up method, which requires almost no higher level knowledge. This is one of the biggest advantages compared to current state-of-the-art.

One of the main obstacles in this field of robotic science is acquiring these symbolic descriptors from sensor data. This poses a critical problem in two respects. First, it is needed for logic-based reasoning and the resulting descriptors form the basis for further learning. Second, it is essential for higher-level planning. The planning component uses the resulting descriptors instead of sensor data. There are numerous ways of bridging the signal-to-symbol gap. First, one needs to define how a logical sequence of sensor signals can be divided into actions and subactions, where each part may be connected to a logical symbol. Second, this must happen in an automatic manner.

Planning and learning usually happens in the symbolic domain. A discrete state space, including discrete actions, is assumed. Throughout the years, a lot of progress has been achieved and various complex domain description languages, such as STRIPS [43], PDDL [77], HAL [76], or ADL [90], have been developed to meet the increasingly demanding AI tasks. Various approaches exist to solve discrete problems in the defined domain [64] and even high level frameworks exist for easy implementation [2, 28].

Thus, more elaborate decision making systems have been proposed to integrate the motion planning more tightly into the action planning domain. In [54, 57, 93] a forward-search planner is used. The task plan is built and the feasibility is continuously checked by a geometric/low level motion planner and being replanned in cases of error. With this motivation in [23, 41], an interface is introduced, which provides "external predicates". These functions are used in the action domain description for checking the feasibility of a primitive action by a motion planner [39]. In [41] the planning language PDDL [77] is extended with so called semantic attachments and the motion planner is changed accordingly.

It has been shown that there is a fine line when taking low level information, i.e. geometric features into account [39]. Obviously, some information is needed for action execution and while a task often can be solved this way, the resulting plan might be inefficient or unfeasible. Taking some high level information into account, a plan made up from low level information can often be enhanced [39]. In this work, a novel approach to planning on a very low abstraction level is introduced: First, it is shown that even without high level knowledge a very wide variety of tasks on that level can be solved. Second, this low level of abstraction can be utilized to retrieve a motor signal in a straightforward way, but also can access the higher level symbolic domain easily. This means, this planner can be used as a way to bridge the signal-to-symbol gap. Therefore, a third abstraction layer between the motion planner and a high level symbolic planner is employed and thus combines the advantages of low- and high level planning. Again, actions are represented by Semantic Event Chains. In this work, it is shown that using SECs and further enriching those by pre- and postconditions, an already very powerful action decision framework is built. This way the framework does not depend on high level symbolic knowledge, but is rather built using a bottom-up method.

3.2. Methods

In this section, approaches to planning are introduced. An overview is given in Fig. 3.2: First, a) the scene is recorded as RGB-D image by an Asus Xtion Pro camera and a Nikon high resolution DSLR. Each depth frame is b) preprocessed by computer vision algorithms. They are segmented via the Locally Convex Connected Patches (LCCP) algorithm [116] into clusters, where each cluster correlates to one object candidate. For object recognition [109] is used; objects are tracked via [88]. All parts of the algorithms are integrated as modular ROS nodes [94]. Next, the objects are categorized based on c) an action ontology, Sec. 3.2.1. This categorization is used throughout the rest of this thesis. Based on this information, a graph structure is retrieved in d) Semantic Event Chain

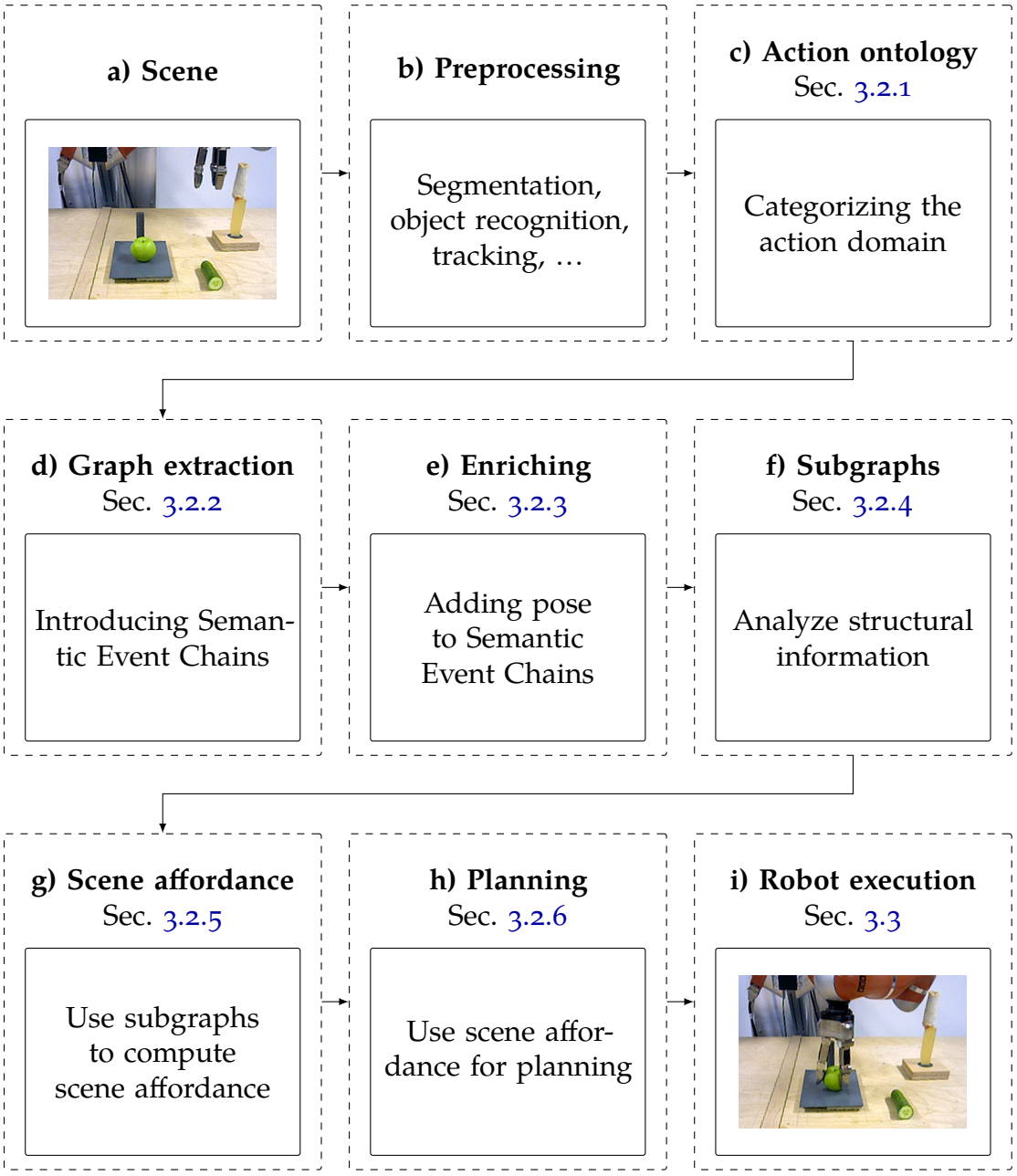


Figure 3.2.: Flowchart of the methods in this chapter and how they relate. Details are explained in Sec. 3.2.

(Sec. 3.2.2). As first contribution, the graph is enriched with pose information by e) a novel 3d reasoning algorithm (Sec. 3.2.3). Afterwards, it is shown in f) that each graph can be reduced to only three different subgraphs (Sec. 3.2.4). This in turn is used to compute g) a scene's affordance (Sec. 3.2.5). Here, for example the question is asked: "Given a tomato, a knife, and a cutting board, what can one do with these objects?" In h), it is shown how the approach can be extended for complex action planning (Sec. 3.2.6). As an example it is analyzed what actions a robot needs to perform when given the command "Make me a sandwich."

3.2.1. Action categories

In this section, we will first define the action domain. Here, the focus lies on actions involving hands and objects (and thus not gestures, etc.). Reasoning about actions seemed for a long time of purely philosophical interest and a detailed review can be found in [15]. The same author states the major viewpoints [133] on action ontologies as: [14, p. 195]

> "Perhaps the most controversial aspect of so called action theory is its subject matter. This subject matter is generally said to be (or to concern) actions, but different philosophers conceive of actions in radically different ways. For some philosophers actions are abstract entities – states of affairs, propositions, sets, or even ordered pairs of some kind. For others, actions are distinctively concrete entities located in space and time. Another group of philosophers, among whom I include myself, have even denied that actions are required for a reasonable action theory, insisting that agents or actors will suffice as the theory's sole objects."

However, apart from the entity concept of an action, one needs to ask the question: "What is an object?" Originally, this subject was perceived as bottom-up and being object-driven. This means different objects allow for different actions. This is the so-called affordance principle [112].

As stated in [133] agency on the other hand suggests that the intended action lets the agent seek for appropriate objects. This point of view leads away from the question "What can you do with all the things in the world", but rather points to the question of "What can you do with your hands?" and recent concepts suggest that objects and actions are intertwined [125]. In Wörgötter et al. [133] it is shown that based on touching relations one can separate actions naturally. This concept is introduced in the following sections in more detail.

Also stemming from this concept in [133] an action ontology is derived, which assumes 26 atomic one-handed actions as shown in Tab. 3.1. Every sequence of actions, e.g. "make a sandwich", can be broken down into a sequence of atomic actions:

1. *Pick & Place* the bread from the table on the cutting board,
2. *Cut* the bread,
3. *Scoop* marmelade,
4. *Put* marmelade *on top* of the bread.

Each action is categorized into three types and each type into two goal categories:

1. Hand-only actions:
    - Rearrange (e.g. hit, push, stir),
    - Destroy (e.g. cut, draw, scoop),
2. Separation actions:
    - Take-down (e.g. take-down, push apart)
    - Break (e.g. rip off, uncover by pick & place),
3. Release determined actions:
    - Construct (e.g. put on top, push together),

Nr	Type	Goal	Instantiation	Example
1	1	r	Punch/hit	with your hand an object
2	1	r	Flick	with your finger nail, quickly
3	1	r	Poke	with your finger tip, slowly
4	1	d	Chop	quickly, with the edge of your hand
5	1	r	Turn = bore (rotate wrist x)	a hole with your finger or your hand
6	1	d	Cut	slowly, with the edge of your hand
7	1	d/r	Scratch	with your finger nail
8	1	d	Scissor-cut/pinch	between your fingers
9	1	d/r	Squash, squeeze	inside your fist
10	1	d	Draw	with finger in sand
11	1	r	Push/pull-without-grasp	regular push, hook-pull, adduct with finger
12	1	r	Stir	with finger
13	1	r	Knead	kneading dough, etc.
14	1	r	Rub/massage	with your hand someone else's body
15	1	r	Lever (rotate wrist y)	e.g. break open a hole
16	1	d	Scoop/ladle	fill your hand with liquid
17	2	t	Take Down or Pick apart	one block from a laterally connected group or a pile by pick & place
18	2	t	Push down or push apart	one block from a laterally connected group or a pile by pushing
19	2	b	Rip off	Rip a piece off an object
20	2	b	Break off	Break a piece off an object
21	2	b	Uncover by pick & place	Pick off an object to uncover another object
22	2	b	Uncover by pushing	Push off an object to uncover another object
23	3	c	Put on top or Put together	two blocks on top of each other or side by side by pick & place
24	3	c	Push on top or push together	two blocks on top of each other or side by side by pushing
25	3	h	Put over	Put one object above another one to cover it completely
26	3	h	Push over	Push one object above another one to cover it completely

Table 3.1.: List of atomic actions as taken from [133]. More actions are listed as "Some (sic) dynamic versions of 17 – 26"; for example, the action "throw-in". According to [133] there are three different manipulation types (listed in the "Type" column): 1: Hand-only-actions; 2: Separation actions; 3: Release determined actions. Abbreviations in the "Goal" column are defined as follows: d: destroying; r: rearranging; c: constructing; t: taking-down; h: hiding; and b: breaking.

- Hide (e.g. put over, push over),

where actions from each goal category share similar trajectories. Next, one needs to define object roles. These roles are determined by the changes that occur following an action in the relation of an object to other objects. An action involves at least two objects: a *hand* and a *main* object. The resulting object list (*hand*, *main*, *primary*, *secondary*, etc.) and their abstract roles are as followed (taken from Reich, Aein, and Wörgötter [97]):

- *Hand* (The object that performs the action): not touching anything at the beginning and the end of the action. It touches at least one object during the manipulation.

- *Main* (The object which is directly in contact with the hand): not touching the hand at the beginning and the end of action. It touches the hand at least once during the manipulation.

- *Primary* (The object from which the *main* object separates): initially touches the *main* object. Changes its relation to not-touching during the action.

- *Secondary* (The object to which the *main* object joins): initially does not touch the *main* object. Changes its relation to touching during the action.

- *Load* (The object which is indirectly manipulated): does not touch the hand. This object touches/ untouches the *main* and untouches/ touches the *container* during the action .

- *Container* (The object whose relation with *load* changes and which is not the *main* object): touches or untouches the *load* object.

- *Main support* (The object on which the *main* object is located): touching the *main* object at least once.

- *Primary support* (The object on which the *primary* object is located): touching the *primary* object at least once.

- *Secondary support* (The object on which the *secondary* is located): touching the *secondary* object at least once.
- *Tool* (The object which is used by the hand to enhance the quality of some actions): touching the hand all the time.

When looking at object roles a different categorization of actions comes to mind: One can define action categories based upon the objects, which the hand interacts with. These fall into three classes:

1. Actions with *main support*: In this category the *main* object is always in touch with the *main support*; an example is shown in Fig. 3.3a.
2. Actions without *main support*: In this category the *main* object is lifted from the *primary* object; an example is shown in Fig. 3.3b.
3. Actions with *load* and *container*: In this category a *container* with *load*, e.g. a glass filled with water, is used; an example is shown in Fig. 3.3c.

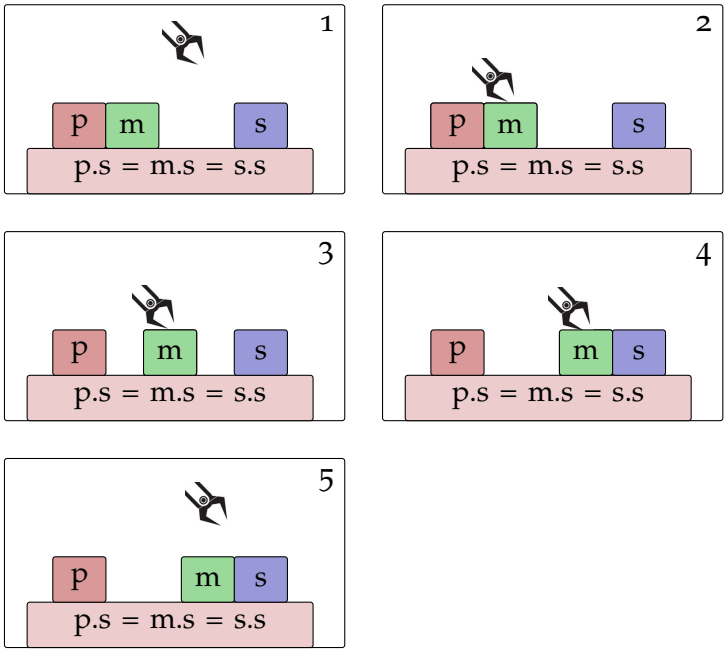
A detailed list of actions is shown in Tab. 3.2. While the first categorization places its focus on the high level goal of the action, the second one is derived from the bottom-up point of view of an object's structural role. In the following sections we will see that the structural role is also important for affordance and planning.

3.2.2. Semantic Event Chains

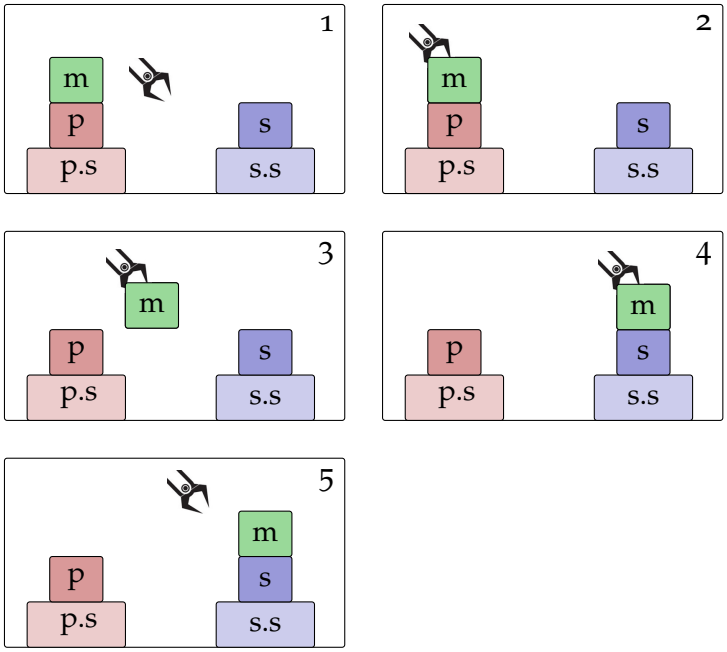
One of the central goals of early human development is to recognize, learn, and lastly imitate actions. Much in life is learned by imitation. For example, a baby might look at her parents walking and try to imitate this behavior. On the other side, a child might learn through unconscious imitation moral codes of society through the conduct of the parents, teachers, movies, or literature. Another way of learning is by trial-and-error. This method is used if no ready-made solution of a specific problem is available. The learner performs random activities until the goal is reached accidentally. A good example is given by younger children

Category	Sub-Category	Example Actions
Actions with main support	Actions with hand, main, and main support	push, punch, flick
	Actions with hand, main, main support, and primary	push apart, cut, chop
	Actions with hand, main, main support, and secondary	push together
	Actions with hand, main, main support, primary, and secondary	push from a to b
Actions without main support (These action have primary, secondary and their supports)	primary $\neq$ secondary and primary support $\neq$ secondary support	pick and place, break off
	primary $\neq$ secondary and primary support $=$ secondary support	pick and place, break off
	primary $\neq$ secondary and primary $=$ secondary support	put on top
	primary $\neq$ secondary and primary support $=$ secondary	pick apart
	primary $=$ secondary	pick and place, break off
Actions with load and container	The relation of load and main changes from N to T (loading)	Pipetting
	The relation of load and main changes from T to N (unloading)	Pour, Drop

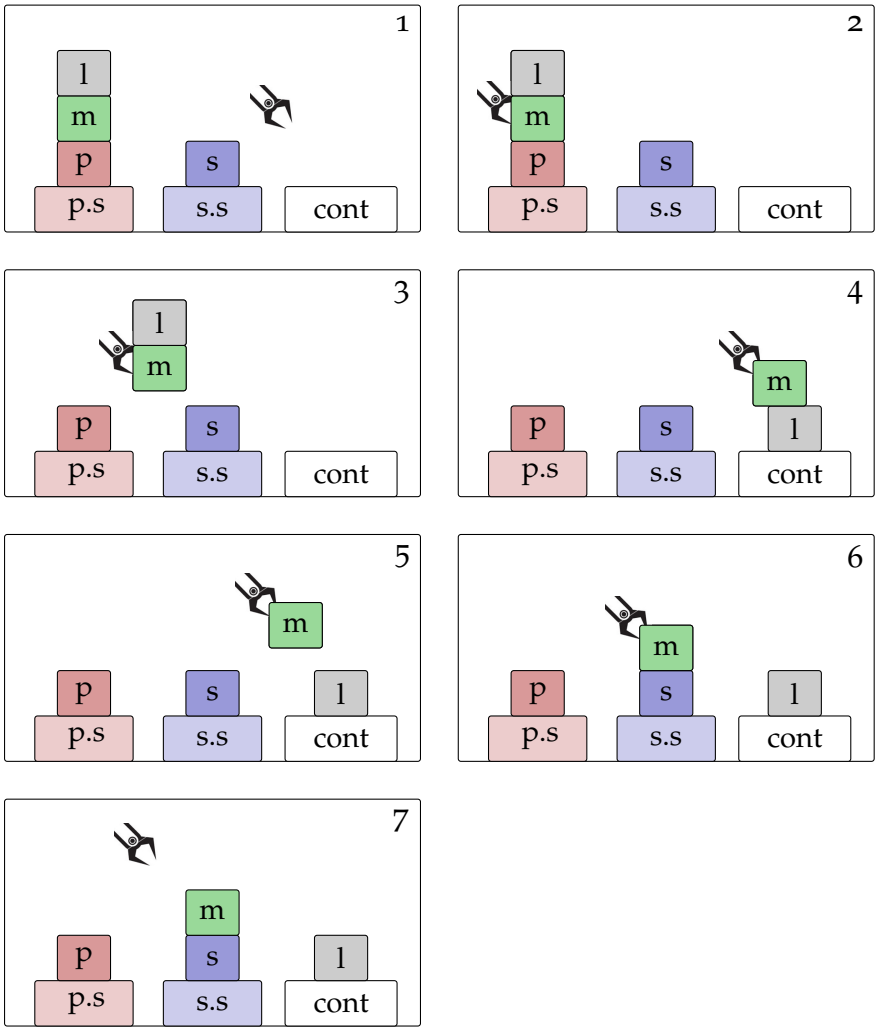
Table 3.2.: Summary of ontology of actions. Actions are divided into three categories and further into sub-categories. There can be more than one action in each sub-category. Taken from Reich, Aein, and Wörgötter [97].



(a) Example action with *main support*: *Pushing*.



(b) Example action without *main support*: *Pick and place*.



(c) Example action with *load* and *container*: *Unloading*. The *main* object m could be, for example, a glass and the *load* l could be water. The water is unloaded into a flower pot.

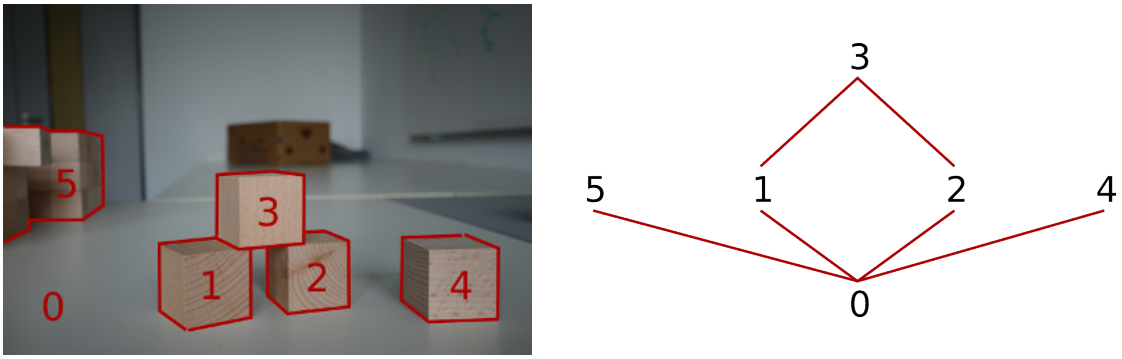
Figure 3.3.: Schematic example actions in the ontology are shown for the three categories. From each category only one action is shown. The objects are marked using the following convention: h = hand, m = main m.s = main support, p = primary, p.s = primary support, s = secondary, s.s = secondary support, l = load, and cont = container (taken from Reich, Aein, and Wörgötter [97]).

playing with wooden building blocks. This type of undirected playing teaches to build small structures, e.g. towers, and thus enable to learn physical properties. Lastly, humans can learn by insight, which is the Gestalt view point. According to this theory, solutions to a peculiar problem may appear sudden. Because this type of learning does not consume much time, it is very important in the education field. As example may serve a jar full of candy, sitting on a kitchen counter. A child wants to reach the jar, but is too small. The kid may sit down, think about the situation and come to the conclusion, that a chair can be used to reach the candy.

Similar to human beings, in cognitive robotics one of the central goals remains to recognize, learn, and lastly imitate actions. However, it has been long addressed that naive observation and raw copying does not suffice to successfully perform an action by a robot [20]. If a human watches a pushing action, for example a pen pushed by a hand, he can bootstrap easily the essence of the action: The goal is to push the pen. This action can be learned and repeated easily. Here, it does not matter, if the left hand, right hand, or a tool is used, the essence is always still captured. Even changes in trajectory or velocity can easily be applied. It is suspected that the mirror-neuron system is involved in this feat; currently, it is not understood how newborns learn these advanced motor skills [101].

For a robot however, it is even today difficult to tell, if a trajectory or a specific object is important to reach a certain goal. This level of invariance is learned by human beings by relating actions with objects and which we call action understanding. In [7] a unified framework is introduced, which enables robots to classify, learn, and repeat actions. This framework is called Semantic Event Chains (SECs). A more detailed view is provided in [6]. Semantic Event Chains store actions as a series of touching and non-touching events.

One example is given in Fig. 3.4a: Some wooden blocks are distributed on a table. First, computer vision clusters, segments, and classifies the objects. For visual purposes these steps were performed manually here. We assume knowledge about "up" and "down" and say that the table is always below the objects. This allows to extract a graph representation as shown in Fig. 3.4b. In this graph



(a) Manually segmented example scene of blocks on a table.

(b) Extracted graph representation of the scene.

Figure 3.4.: A visualization of an object graph. Computer vision identifies and separates objects and their relative structure to each other (left image). One Semantic Event Graph (right image) results directly from the structure. Please note that multiple roots for one graph are allowed.

there is the representation for *touching* — as seen in the example in the relation between block 1 and 3 — and *not touching* — as seen in block 1 and 4. The graph is undirected, unidirectional, and each edge is of unit length. Thus, the same graph can be represented in a symmetrical adjacency matrix as follows

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & \mathrm{T} & \mathrm{T} & \mathrm{N} & \mathrm{T} & \mathrm{T} \\ \mathrm{T} & 1 & \mathrm{N} & \mathrm{T} & \mathrm{N} & \mathrm{N} \\ \mathrm{T} & \mathrm{N} & 2 & \mathrm{T} & \mathrm{N} & \mathrm{N} \\ \mathrm{N} & \mathrm{T} & \mathrm{T} & 3 & \mathrm{N} & \mathrm{N} \\ \mathrm{T} & \mathrm{N} & \mathrm{N} & \mathrm{N} & 4 & \mathrm{N} \\ \mathrm{T} & \mathrm{N} & \mathrm{N} & \mathrm{N} & \mathrm{N} & 5 \end{pmatrix} \end{array}, \tag{3.1}$$

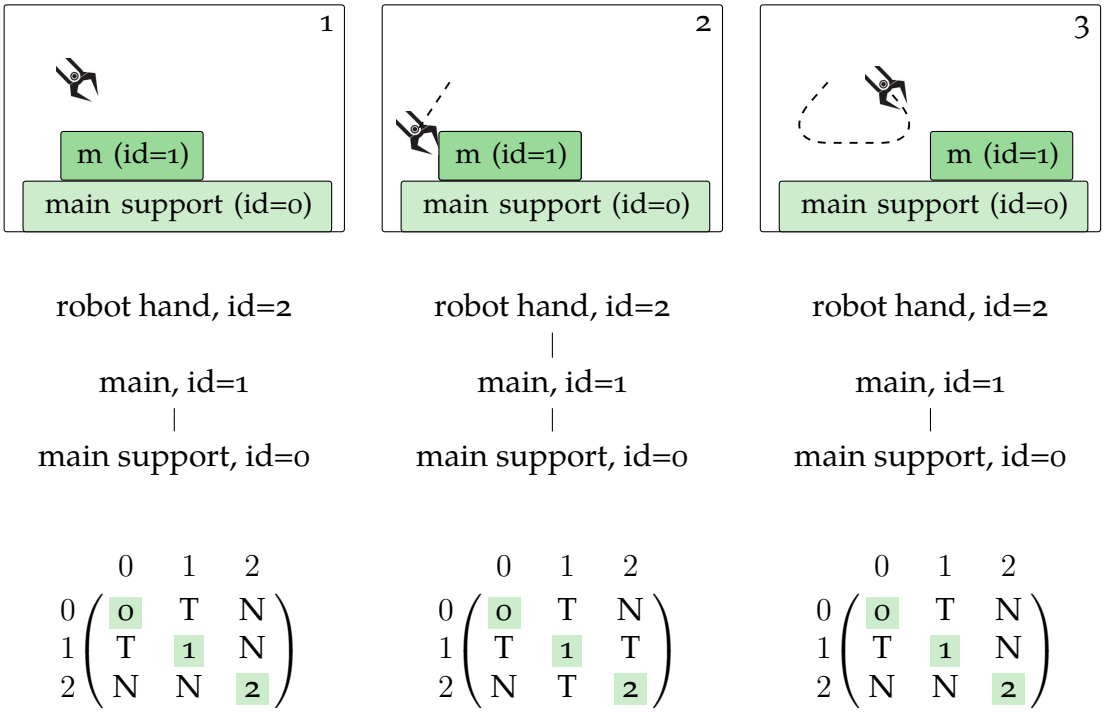
where "T" marks an edge and "N" stands for a not-touching relation. This matrix is called Semantic Event Matrix (SEM). It is symmetric and contains the ob-

ject identifiers on the diagonal (marked in green). Additionally to touching and not-touching, there may be an edge named *absent* "A", which is used when new objects come into the scene; for example when a cucumber is being cut into two pieces, or when an object is uncovered during a scene. Now, each change in the scene corresponds to a change in the touching relation and therefore results in a new matrix. A list of SEMs is called a Semantic Event Chain. The camera frame, in which the change of relation occurs is called a keyframe. SECs are independent of the time domain and the robotic hardware.

A second (and also artificial) example of a *pushing* action is shown in Fig. 3.5. A robot pushes the *main* object "m" along its support. In the first keyframe, there is only one touching connection between the object and its support, which is shown in the graph below the pictogram and also reflected in the SEM below the graph. Next, the robot begins to touch the *main* object and holds that connection for the entire trajectory, i.e. for a longer period of time. The last (third) keyframe is generated as soon as the robot looses contact to the *main* object.

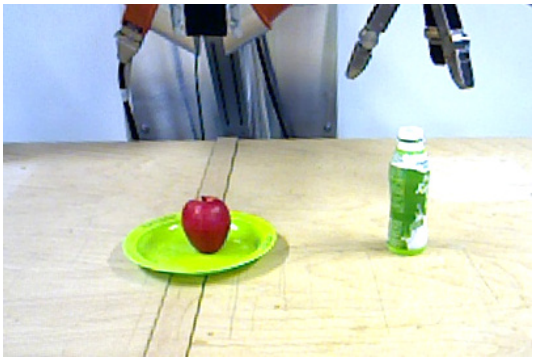
A third example of a *pick and place* action is displayed in Fig. 3.6; it is derived from a real world experiment. In the first keyframe an apple is on top of a plate and the robot hand hovers above the table. In the next keyframes it holds the apple, lifts it off the plate, and places it on the table. This sequence of touching/non-touching relations is unique for this type of action. If in this scene the robot were to push the apple on the plate, the graph sequence would look differently. While the first two keyframes would look like the pick-and-place example, the third keyframe would be equal to the first: The robot hand hovering in the air. Immediately one problem becomes apparent: In both examples the first two keyframes are alike, even though in one example the apple is picked and in another example it is pushed. Therefore, reliable action recognition is only possible on a very late stage of action execution.

Another problem surfaces, when thinking about the *scratching* action: i.e. scratching with a robot finger on paper. This action is indistinguishable from the pushing action on the SEC domain, they only differ in the robot hands trajectory. Both actions last for three keyframes where the first and last keyframes are identical;

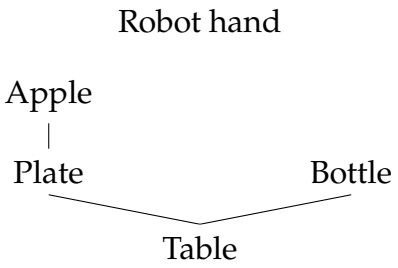


$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & T & N \\ T & 1 & N \\ N & N & 2 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & T & N \\ T & 1 & T \\ N & T & 2 \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & T & N \\ T & 1 & N \\ N & N & 2 \end{pmatrix} \end{array}$$

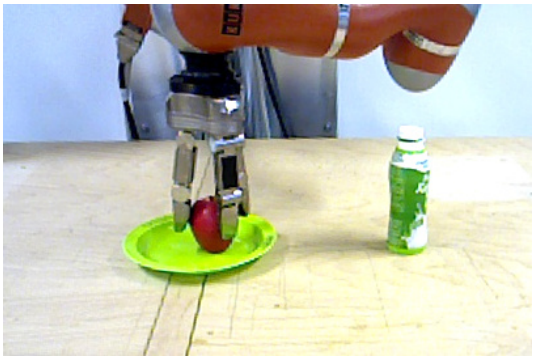
Figure 3.5.: An example showing a pushing action in the SEC domain. The first row shows a pictogram view of the action. The *main* object, denoted with "m", sits on top the "main support" and the robot is not touching the *main* object. In the second keyframe the robot touches the *main* object and pushes it to the right. The robot's trajectory is marked with a dashed line. However, this trajectory information is not encoded in the SEC. In the third keyframe the robot hand is removed from the *main* object. The middle row holds a graph representation of the touching and not-touching relations; touching relations are marked with a line. In the bottom row the graph is represented as SEMs. All three matrices hold a lot of static information. Therefore, a short form, which removes all static information, is introduced. For this example one could also write: "main object – robot hand: N T N".



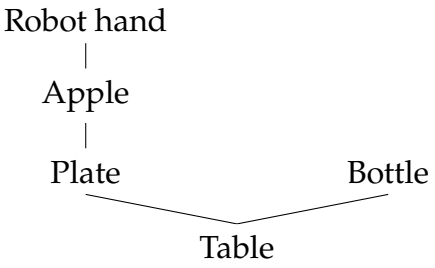
(a) Keyframe 1, this is the very first camera frame.



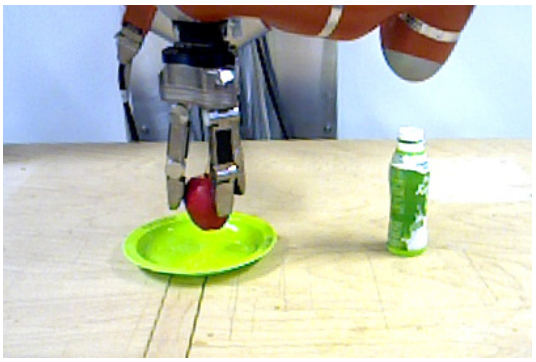
(b) Graph representation of Keyframe 1.



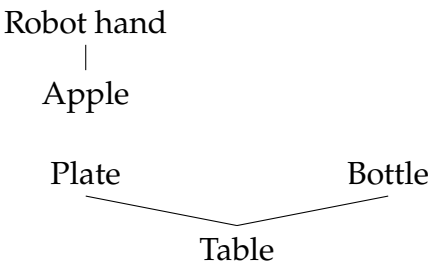
(c) Keyframe 2, the robot arm touches the apple.



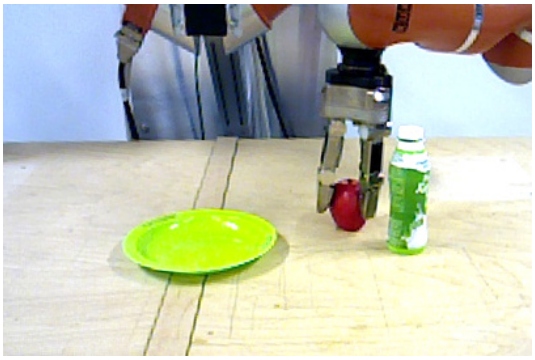
(d) Graph representation of Keyframe 2.



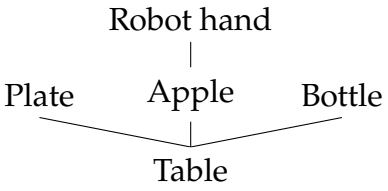
(e) Keyframe 3, the robot lifts the apple of the plate.



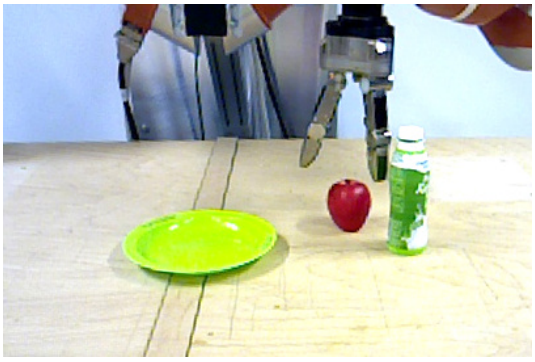
(f) Graph representation of Keyframe 3.



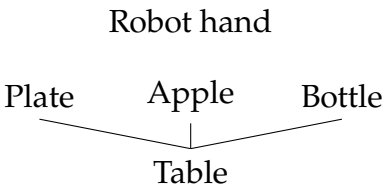
(g) Keyframe 4, the apple is placed on the table.



(h) Graph representation of Keyframe 4.



(i) Keyframe 5, the robot arm lets go of the object.



(j) Graph representation of Keyframe 5.

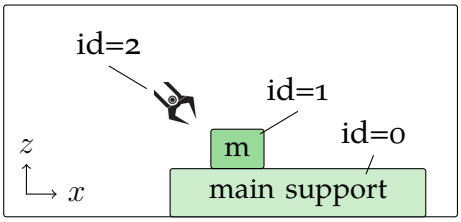
Figure 3.6.: Frames from a robot demonstration: The robot picks an apple from a plate and places it on the table. The corresponding graph representation is given on the right side.

in the second keyframe the robot touches the object. Here, it is important to note that SECs do not store trajectory or time information. While on the one side this seems like a disadvantage, it is in fact the biggest strength of the SEC domain. Trajectory information is hardware dependent and therefore robot specific. Using SECs one can recognize [5], learn [3, 7], and repeat [3] a very wide range of actions. Furthermore, actions can be enriched using action-related trajectory information and thus be used for bootstrap learning [5].

Still, not necessarily all actions a human can perform may be encoded using SECs. For some actions higher level knowledge is needed. As an example of these actions, *spreading the marmelade* may serve as an example. It first depends on the trajectory's velocity and second on the size of the bread, which cannot be described based on SECs alone. To overcome some of these shortcomings of SECs, in the next chapter pose information is added. This will allow for a framework to execute actions, which generalizes while still being independent of the time domain.

3.2.3. Enriched Semantic Event Chains

As seen in the last chapter, for robot execution of Semantic Event Chains, some additional pose information must be included. Therefore, SECs are enriched by higher level structural information. Each entry of one keyframe matrix now may not only hold "T", "N", or "A", but also an additional vector containing the relative pose between the two objects. An example can be seen in Fig. 3.7, which is similar to the first keyframe in the example Fig. 3.5, but introduces an additional coordinate system. Following this approach, the Semantic Event Chain turns into an Enriched Semantic Event Chain (ESEC) as



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & \mathrm{T} & \mathrm{N} \\ \mathrm{T} & 1 & \mathrm{N} \\ \mathrm{N} & \mathrm{N} & 2 \end{pmatrix} \end{array}$$

Figure 3.7.: This is the same scene as shown in Fig. 3.5 — a robot pushing the *main* object along its support. Please note the coordinate system, which is used when the keyframe matrix on the right is enriched by relative pose information, see Eqn. (3.2). For clarity only two dimensions are used here (where $y = 0$).

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & (\mathrm{T},\ (0,\ 0,\ 1)) & (\mathrm{N},\ (0,\ 0,\ 1)) \\ (\mathrm{T},\ (0,\ 0,\ -1)) & 1 & (\mathrm{N},\ (-0.71,\ 0,\ 0.71)) \\ (\mathrm{N},\ (0,\ 0,\ -1)) & (\mathrm{N},\ (0.71,\ 0,\ -0.71)) & 2 \end{pmatrix} \end{array}, \tag{3.2}$$

where the ID is marked in green. A more general form for the keyframe matrix $\underline{K}$ is

$$\underline{K} := \begin{pmatrix} a_{0,0} & \vec{a}_{1,0} & \cdots & \vec{a}_{i,0} & \cdots & \vec{a}_{N,0} \\ \vec{a}_{0,1} & \ddots & & & & \vdots \\ \vdots & & & & & \\ \vec{a}_{0,j} & & & & & \\ \vdots & & & & & \\ \vec{a}_{0,N} & \cdots & & & & a_{N,N} \end{pmatrix} \tag{3.3}$$

where

$$a_{i,i} := i \tag{3.4}$$

and for $i \neq j$

$$\vec{a}_{i,j} := (r,\ \vec{p}_0,\ \cdots,\ \vec{p}_m,\ \cdots,\ \vec{p}_k)\,. \tag{3.5}$$

The scalar entries "T", "N", and "A" are replaced by vectors of the form $(r,\ \vec{p}_m)$, where relation $r \in \{\mathrm{T},\ \mathrm{N},\ \mathrm{A}\}$ and $\vec{p}_m \in \Re^d$. The vector $\vec{p}_m$ is normalized to unit length such that $|\vec{p}_m| \stackrel{!}{=} 1$.

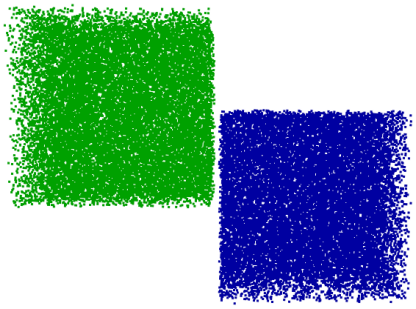
One might expect that one vector is enough to define the relative pose between two objects. However, using only one vector often does not catch the shape of the other object. For example, one might imagine a simple cube, surrounded by an L-shaped block. Here, at least two vectors are needed to describe the directions, in which the cube must not be moved in order to not touch the L-shaped block. In these experiments, as will be shown below, up to $k = 8$ different pose vectors per relation are allowed. Below, an algorithm for 3d geometrical reasoning is devised, which will output these vectors. Parts of this section have been published in Reich, Aein, and Wörgötter [97].

First, please imagine a *pick & place* action. It is quite difficult to determine the "correct" rotation of the hand to pick up an object. Additionally, if the scene is cluttered, it is often hard to tell in which direction the hand should move to lift the object and to not touch any other objects. Here, a geometrical reasoning algorithm is needed, which will output this information. As already mentioned, there is — due to Semantic Event Chains — a good understanding of top and bottom of the scene. However, the question which objects are left and right of the *main* object remain. Most certainly the robot will also encounter objects of different sizes and shapes, which must be taken into account, too.

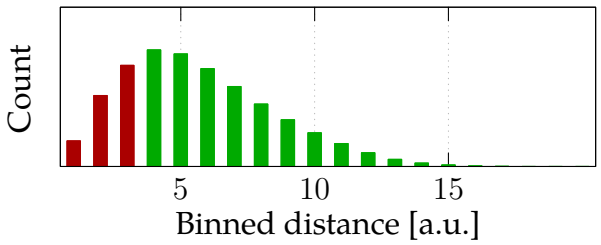
Currently, the vast field of different robot hardware and use cases has shown that geometric reasoning algorithms do not generalize well. For example, in [55] the focus lies on computing efficiently the relative position of multiple moving objects to each other; results are shown only in simulation. In [40] geometric reasoning is combined with a causal symbolic planner, which in turn is used in an industrial use case. On a more abstract layer, [75] defines attention space using low level geometric features. Thresholds define a space around the robot for human-robot collaboration. In [65, 117, 118] geometric reasoning is used to infer information about local structures. These are used in medical robots [117, 118].

As of today there is no labeled benchmark for geometric reasoning in real world scenarios. Thus quantitative evaluation is not very meaningful as one would need to generate the ground truth first. This, however, is hardware specific, robot dependent, and does not generalize.

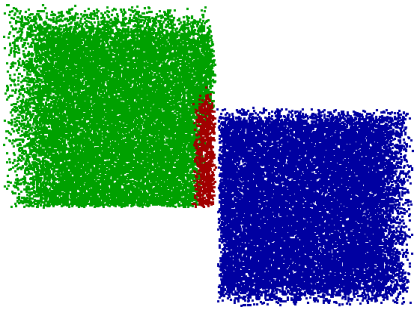
A step-by step explanation is shown in Fig. 3.8. For visualization purposes the relative position of two cubes to each other is analyzed: one green and one blue. In a very simple approach, one could reduce the objects to one point in space, for example its mean or average position. This, however, will ignore object sizes as well as shapes. Instead, a more general solution, which does not depend on object size, shape, or distance is sought. First, the distance from each voxel from one cube to each voxel in the other cube is computed and binned as shown in Fig. 3.8a and Fig. 3.8b. For two symmetrical objects a Poisson shaped distribution is to be expected. All voxels, which are below the first maximum and belong to the green cube are taken into consideration; these points are marked red in the histogram. The corresponding voxels are marked in Fig. 3.8c in red, too. Next, the normals of these voxels are calculated. These normals will, as per definition, point away from the green cube and will always point towards the blue cube. These normals are clustered using a k-means clustering algorithms. While undersegmentation will be harmful — as not all directions are found — but oversegmentation is not, a k that is greater than the expected number of directions is used. It was found heuristically that $k \approx 8$ leads to good results for most real-world examples. Lastly, a half sphere around each resulting cluster is spawned



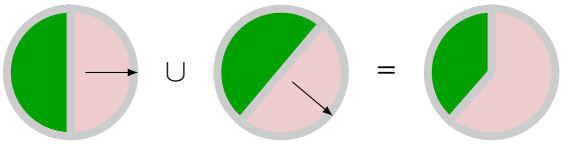
(a) Two blocks serve as an example for geometric reasoning. Here, one is interested in which direction the green cube can be moved without touching the blue one.



(b) First, the distances from all voxels of the green block to all voxels from the blue block are binned. In the next step all voxels of the green block, which are below the maximum of the histogram (marked in red) are taken into account.



(c) The voxels found in Fig. 3.8b are marked in red. Normals of these voxels are computed and k-means clustering of the normals is performed (here: $k = 2$).



(d) A half sphere around each of the $k = 2$ resulting vectors is spawned (here: half circle for visualization) and the union of all spheres is computed. The union, above marked in red, marks the "forbidden" directions.

Figure 3.8.: Step-by-step explanation of the geometric reasoning algorithm.

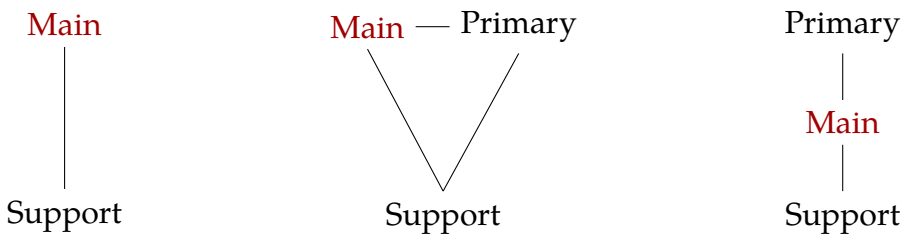
(half circle in 2d as shown in the example in Fig. 3.8d). The union of all spheres points to the blocked directions, which is marked in red in the example — the direction where the blue cube is located at. This computation is performed for each object, which is in a certain radius around the *main* object. The radius is hardware dependent and defined by how much space the robot hand needs to safely grasp or push an object.

One could now devise a library of actions: Each action is combined with a robot-hardware specific set of trajectories. Parameters, which are needed for execution, i.e. where an object is located or from what angle it is safe to approach, can be generated in an automated manner by the here proposed geometric reasoning algorithm. One such library for the Kuka Lightweight Robot was developed in collaboration with this work and published in [1]. This library will be used during the next sections for action execution. However, before one proceeds to the execution phase, one needs to analyze the structural information of ESECs in more detail.

3.2.4. Structural information

Semantic Event Chains offer a good understanding of top or bottom and the now introduced Enriched Semantic Event Chains additionally express "left" and "right". This gives valuable information on how to execute an action, but very little information, whether an action is feasible. In this chapter structural topologies of object graphs are analyzed. For visualization, we will assume in this chapter that object segmentation and recognition performs in a perfect manner. Furthermore, one must assume that the support of an object is always below the object itself and that the object recognition software provides knowledge about the supporting objects, e.g. if a table is present, the table is marked as supporting object. This also marks the constraints of this approach: Hanging structures are not analyzed here.

As noted in Sec. 3.2.1, an action will always be performed on the *main* object. To analyze the structure around the *main* object, subgraphs are created, where



(a) Example: Plate on a table.

(b) Example: Plate next to a cup on a table.

(c) Example: Apple on a plate on a table.

Figure 3.9.: Only these three subgraphs may exist around the *main* object. Any graph structure, which contains at least a *main* object and its support, can be reduced to a series of these subgraphs. Any subgraph consists of the *main* object, its support, and up to one more object.

one subgraph always holds the support and the *main* object, and up to one other object, which is called *primary* object. Remarkably, there are only three possible topological subgraphs to which all scenes that include the *main* object can be reduced. These three possible cases are shown in Fig. 3.9:

1. The *main* object has only one touching relation. The touched object is a support (see Fig. 3.9a). A real world example could be a plate lying on a table.

2. The *main* object has two touching relations. One is a support, the second one is another object, which is also touching a support (see Fig. 3.9b). A real world scene could be a plate and a cup on a table, both are touching each other.

3. The *main* object has two touching relations. One is a support, the second one is another object, which does not touch a support (see Fig. 3.9c). Lastly, this structure could emerge from a plate with an apple on top; the plate being the *main* object here.

Next, a real-world example as shown in Fig. 3.10a is analyzed. The computer vision system delivers a low level graph structure, but also two pieces of high level information: which object is the table/the support and what is the *main*

object. The supporting object is important, because it is always below all other objects. The objects are named analogously to Sec. 3.2.1, meaning the manipulated object is called *main* object. The resulting graph is decomposed into two subgraphs: one tower-like structure and one structure where the *primary* is situated next to the *main* object. SECs do not offer information about size, and the bottle O_1 could be far away of the plate – it is therefore disregarded.

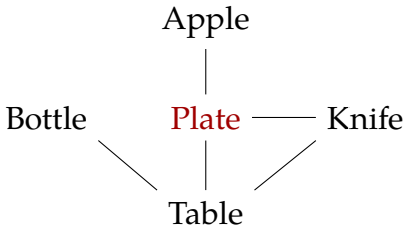
Now, the complex graph structure is broken down to much simpler subgraphs. It is already well known that the mere existence of objects already suggests scenes [138]: For example, an image of a plate, cutlery, a roll, and a glass of juice will more likely show a breakfast scene, than an image of a supermarket. But an isle inside a supermarket could show the same objects from the breakfast scene (and probably many more at the same time) in a different context. One can easily see, that a scene's affordance heavily depends on discriminative spatial layouts [68, 121]. In the next section, it is analyzed how the subgraphs can be used as additional features to ESECs to compute a scene's affordance.

3.2.5. Affordance of Semantic Event Chains

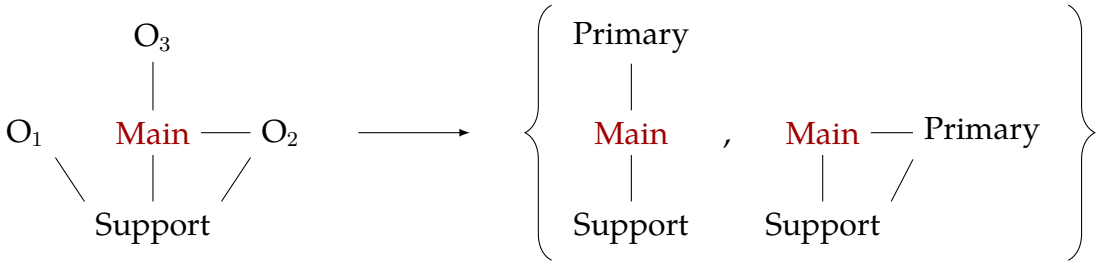
Contrary to intuition, the very simple form of Semantic Event Chains to store action sequences still holds a lot of information about the structure of the scene. However, this information mostly regards top-to-bottom structures, while "What is right?" or "What is left?" is not encoded. This was fixed by enriching SECs with additional pose information. Then, the resulting graph structure was decomposed into three different types of subgraphs. Based on this information, we will look at the scene affordance and combine it with a set of movement primitives as introduced by [1]. Parts of this chapter have been published in Reich, Aein, and Wörgötter [97].

From everyday life we know that different scenes suggest different actions, e.g. a board, a tomato, and a knife suggests to *cut the tomato with the knife*. However, assessing whether or not a robot could actually do this, whether it should/could do rather something else or whether not much can be done at all given such

(a) The original scene as viewed by the robot. A higher level algorithm (or human) chooses the plate as *main* object to manipulate



(b) The extracted graph relations after object classification. Plate is recognized as *main* object.



(c) The abstract graph is cut into two subgraphs around the *main* object, the plate. As the bottle has no direct connection to the plate, no subgraph is generated. As the subgraph consists of at least the support and *main* object, the third object is automatically named *primary* object, see Sec. 3.2.5. "O" stands for other, not closer defined objects.

Figure 3.10.: A scene, as recorded by a robot is analyzed and a graph structure is generated. As *main* object the plate is chosen by either human or higher level algorithms. For each object around the *main* object a subgraph is generated.

scenes remains a difficult problem. It amounts to estimating the affordance of certain actions given the context provided by the scene. In this section, scene affordance based on Semantic Event Chains will be analyzed.

Many approaches have been evaluated to solve this problem. In [104] a complex network of geometrical relations in the spatial and temporal domains is used. Via Support Vector Machines (SVMs) topological features and symbolic meanings are learned. In [114] patterns of functional relationships are defined, e.g. the object "work surface" with the action "manipulate". Similar, in [67] posture templates are applied to the input data of each frame. The resulting series of templates eventually forms a library of actions. The authors use variable-length Markov models for learning.

Staying closer to the actual motion patterns one can also break down actions into segments, using — for example — Principal Component Analysis (PCA) as in [134]. A motion sequence is projected into a state space, which is then mapped to the first n principal components. In that reduced state space a threshold is applied and the action is divided into two parts. This results in action sequences, which usually end at points of high variance, usually time points of importance. The same is iteratively applied to each subspace until some exit criterion is met. The resulting segments could then be interpreted as meaningful action parts.

There are also non-vision-based methods available; for example in [78] Radio-Frequency Identification (RFID) chips are placed on wrist bands and objects to cluster objects and actions into groups. An interleaved hidden Markov model is used for learning. Another approach uses GPS-based geo information to learn actions, which span a longer time frame, e.g. commuting to work and match those with objects. These methods will not be discussed any further, as the focus lies on vision here.

Nr	Name	main is main of selected action: main — support	main is main of selected action: main — primary — support	main is main of selected action: primary — main — support	main is secondary of selected action: main — support	main is secondary of selected action: main — primary — support	main is secondary of selected action: primary — main — support
1	Punch/hit	✓	✓	✓	n	n	n
2	Flick	✓	✓	✓	n	n	n
3	Poke	✓	✓	✓	n	n	n
4	Chop	✓	✓	-	n	n	n
5	Turn/bore	✓	✓	✓	n	n	n
6	Cut	✓	-	-	✓	✓	-
7	Scratch	✓	✓	✓	n	n	n
8	Scissor-cut/pinch	✓	-	-	n	n	n
9	Squash, squeeze	✓	-	-	n	n	n
10	Draw	✓	-	-	n	n	n
11	Push	✓	✓	-	n	n	n
12	Stir	-	✓	-	n	n	n
13	Knead	✓	✓	-	n	n	n
14	Rub/massage	✓	-	-	n	n	n
15	Lever (rotate wrist y)	✓	✓	✓	n	n	n
16	Scoop	✓	✓	-	n	n	n

17	Take Down	✓	✓	-	n	n	n
18	Push down	✓	✓	-	n	n	n
19	Rip off	✓	-	-	n	n	n
20	Break off	✓	-	-	n	n	n
21	Uncover by pick & place	✓	-	-	n	n	n
22	Uncover by pushing	✓	-	-	n	n	n
23	Put on top	✓	✓	-	✓	✓	-
24	Push on top	✓	✓	-	✓	✓	-
25	Put over	✓	✓	-	✓	-	-
26	Push over	✓	✓	-	✓	-	-
27	Push apart	-	✓	-	✓	✓	✓
28	Push together	✓	-	-	✓	✓	✓
29	Push from x to y	✓	-	-	n	n	n
30	grasp	✓	✓	-	n	n	n

Table 3.3.: List of preconditions for atomic actions on the SEC level (action list as shown in [133]). A "✓" denotes that the structure is allowed, if the action needs to be executed; the actions marked with "-" are not allowed; "n" is used, where the structure is not applicable as the state of the *secondary* is of no relevance. The left three columns show preconditions for the *main* object. The right columns show the preconditions of the *secondary* object of an action. Please note that the action's *secondary* object turns into the *main* object of the subgraph.

All these approaches are problematic as it remains difficult to smoothly link sensor signals (e.g. from scene analysis) to symbolic action concepts and back to the signal domain to create trajectories needed by the robot's actuators. We ask: What is needed to push (or pick, or cut, etc.) a certain object? Which are the general preconditions required for this regardless of the actual objects in the scene? And — if those hold — are also the specific conditions met to actually do it?

The original ESEC framework did not much care about objects. Here, a layered structure on top of the Semantic Event Chains is incorporated, which still allows affordance analysis. This will create an object-action-linked ontology of manipulations, where these object roles define the general preconditions that need to be met to perform a certain action at all.

In this study the robot selects one object in a scene and asks — like a child during play — what could I do with it? The framework will then analyze the situation and suggest possible manipulation actions, thereby addressing the problem of context dependent affordances. A three-layered system is used: During the first layer SEC-based relations are evaluated. The second layer solely analyzes the topological structure of the *main* object, and the third layer consists of a set of movement primitives, which are needed to execute the action.

Layer 1: SEC-based object relations at start. The first layer analyzes whether the first matrix of a Semantic Event Chain is fulfilled. If, and only if these touching relations are not violated, the action could commence. This is not yet sufficient to select actions.

Layer 2: Object Topologies. All actions are always performed at the *main* object and this will only be possible, if the SEC-preconditions hold and if the *main* object appears in the scene with certain topological connections to other objects. To analyze these connections, the nearest neighbors of the *main* object are inspected as shown in Sec. 3.2.4. Using these subgraphs, one can determine the remaining preconditions. For example, a tower structure around the *main* object is not

allowed for a pushing action as the top object could fall down. Now, one can attach a set of allowed structures to each action.

One action may include more than one object, e.g. *push object 1 and object 2 together*, where object 1 is treated as *main* object and object 2 as *secondary*. Of course, the preconditions for the *secondary* object also need checking. A complete list of all preconditions is shown in Tab. 3.3. The left three columns show preconditions for the *main* object. As explained above, subgraphs around the *main* object are created and each subgraph is checked. The right columns show the same for the *secondary* object. Please note that the *secondary* object of the selected action becomes the *main* object of the subgraph.

Layer 3: Movement Primitives. The third layer consists of a set of movement primitives as described in [1]. These movement primitives are hardware dependent. Two such commands shall be explained in more detail and serve as an example: *move(object, transformation T)* and *grasp(object)*. The *move(object, transformation T)* primitive sends a command to the robot to move to a pose which is determined by applying transform T to the pose of $object$. The transformation T contains a translation vector and rotation matrix. *grasp(object)* closes the fingers of the robot hand to securely grasp an object.

For example, when the robot should grasp the *main* object, the *move(main, T)* primitive is performed to move the robot arm end effector to a proper pose for grasping. The translation vector of transformation T will lead the hand to the object, while the rotation part needs to be set such that the robot approaches the *main* object from a proper angle. This is necessary to avoid possible collisions with other objects near the *main*. The parameters for the transformation and an object's pose are taken from ESEC.

For testing the algorithm on real world robot hardware, the processing pipeline is as follows:

1. One object is chosen as *main*. This may happen by user input, or as the output of another, higher level algorithm. Here, a toddler playing and

learning about its environment is simulated. This means a *main* object is chosen randomly and the question is what one can do with it.

2. The complete list of all considered manipulation actions, of which there are 30 (see Tab. 3.1), is loaded.

3. For all of them the first layer of the ontology is used to check whether the *main* object in this scene fulfills its SEC preconditions.

4. For those which passed the first layer successfully, all possible subgraphs around *main* are computed and checked with the second layer of the ontology: The topological preconditions by which the list gets further reduced.

5. Now one can use the third layer and extract the required action primitives from the ontology.

6. This concludes the preparation stage and this information is sent to the execution engine.

3.2.6. Using affordance for planning

The main goal of this work is to execute, if possible, an action based on input as *cut the apple with the knife*. Based on the input a list of actions is to be created, which ultimately executes the requested action. If, for example, in the task *cut the apple with the knife*, the apple is still on the table (and not on the cutting plate), the apple cannot be cut. The simple command *cut the apple* is expanded to a list of atomic actions, which need to be executed first: *pick the apple and place it on the plate*.

Furthermore, a list of atomic actions can be condensed to a single recipe: One example could be *make a fruit salad*. This recipe could include the atomic actions *pick the apple and place it on the plate*, *cut the apple*, *pick and place the apple in the bowl*, and so on. Recipes can be stored and later used to make up even larger tasks: The recipe *make a fruit salad* can be included in the recipe *prepare dinner*.

To reduce the complexity of preconditions any graph can be decomposed into three types of subgraphs (Sec. 3.2.4). The preconditions are analyzed analogously to Sec. 3.2.5 as follows: First, subgraphs for the *main* and, if needed *secondary* object are created. One subgraph consists of the *main*/*secondary* object and one of its directly touching neighbors; this means there can be more than one subgraphs per *main* object. Also, the subgraph contains at least one supporting object (for the sake of simplicity only supporting objects, which are beneath the *main*/*secondary* object, i.e. objects hanging somewhere will not be covered, are analyzed). The maximum number of touching non-support-objects in a subgraph is one, as for each non-support-object a new subgraph is created.

On this reduced level of complexity, the precondition for each action can be easily determined. For example, a tower structure as shown in Fig. 3.9c is not allowed for pushing actions. All preconditions are listed in table Tab. 3.3. Please note that object size, density, shape, and most significantly pose are not considered on this planning stage. It is easy to see that this might lead to problems in the execution phase (e.g. "inside" is hard to define). These physical constraints define the limits of this planning approach.

Again, the ontology as defined in Tab. 3.1 is used. To each atomic action from the ontology a set of allowed subgraph structures is associated. These subgraphs make up the preconditions of an action. An illustration of the action *push* is as follows (cf. Tab. 3.3):

- **Name:** push.
- **Structure 1 for main: No objects are touching the main object.** This corresponds to Fig. 3.9a. For *pushing*, this structure is allowed for execution.
- **Structure 2 for main: Another object around the main object.** Is there another object next to the *main* object, touching the *main* object as well as the support? For *pushing*, this is allowed (see Fig. 3.9b).
- **Structure 3 for main: Another object on top of the main object.** Is there another object on top of the *main* object; this results in a tower like structure as shown in Fig. 3.9c. For *pushing*, this is not allowed.

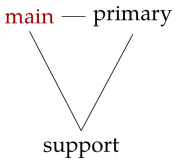
- **Structure 1 for secondary: No objects are touching the secondary object.** No *secondary* object needed for *pushing*.
- **Structure 2 for secondary: Another object around the secondary object.** No *secondary* object needed for *pushing*.
- **Structure 3 for secondary: Another object on top of the secondary object.** No *secondary* object needed for *pushing*.

As the preconditions are known for each action, it can be checked whether an action can be executed. Analogous to the preconditions, to each action one post-condition is defined, i.e. how does the scene look after the action is performed by the robot. An example, again for the *pushing action* is as follows:

- **Name:** push.
- **Structure 1: No objects are touching the primary object.** Are no objects touching the *primary* object after action execution? For *pushing*, this is the expected state.
- **Structure 2: Another object around main object.** A *pushing* cannot result in this state.
- **Structure 3: Another object on main object.** A *pushing* cannot result in a tower structure.

All postconditions are listed in Tab. 3.4. This means, one can predict the outcome of one atomic action on the SEC level. If multiple postconditions are possible, the predicted outcome will be the structure, that is used as input. Several actions may lead to the same desired goal state: For example *pick and place* and *pushing* both may be used to place an object at a different position on top of a table. Thus, as there is no information beforehand which action would be preferable, all actions need to be simulated first.

The integration into the action-perception loop of this system is shown in Fig. 3.11. In a) the perception side is shown: It contains all computer vision steps, which are described in Sec. 3.2. Output of a) is a SEM of the current scene, a list of

Nr	Name	Structure		
				
		cf. Fig. 3.9a	cf. Fig. 3.9b	cf. Fig. 3.9c
1	Punch/hit	✓	✓	✓
2	Flick	✓	✓	✓
3	Poke	✓	✓	✓
4	Chop	✓	✓	-
5	Turn/bore	✓	✓	✓
6	Cut	-	✓	-
7	Scratch	✓	✓	✓
8	Scissor-cut/pinch	✓	-	-
9	Squash, squeeze	✓	-	-
10	Draw	✓	-	-
11	Push	✓	✓	-
12	Stir	-	✓	-
13	Knead	✓	✓	-
14	Rub/massage	✓	-	-
15	Lever (rotate wrist y)	✓	✓	✓
16	Scoop	✓	✓	-
17	Take Down	✓	✓	-
18	Push down	✓	✓	-
19	Rip off	✓	-	-
20	Break off	✓	-	-
21	Uncover by pick & place	✓	-	-
22	Uncover by pushing	✓	-	-
23	Put on top	✓	✓	-
24	Push on top	✓	✓	-
25	Put over	✓	✓	-
26	Push over	✓	✓	-

27	Push apart	✓	-	-
28	Push together	-	✓	-
29	Push from x to y	✓	-	-
30	grasp	✓	✓	-

Table 3.4.: List of postconditions for atomic actions on the SEC level (action list as shown in [133]). A "✓" denotes that the structure is a possible outcome, if the action needs to be executed; the actions marked with "-" are not allowed.

all recognized objects, and labeled point cloud. This data is handed to b) the SEC planner and the geometrical reasoning node. The SEC planner computes a sequence of actions, which is transferred to the c) robot hardware. Here an industrial Kuka Lightweight Robot Arm [18] controlled by Dynamic Movement Primitives (DMP) as described in [1] is used. The entire software of the system is realized in ROS [94], which enables a highly modular system structure.

A schematic overview of the planning algorithm is shown in Fig. 3.12. The semantic relations of the current scene, as analyzed by computer vision algorithms, are used as a) first input. The second input b) is given to the planner in form of a target action, e.g. *pick and place object A on top of object B*. As the action has defined preconditions, the goal state can be derived from the current scene in form of a second SEM. The aim of the planner is now to find a set of ordered actions, which will transfer the current scene to the goal state using a Semantic Event Chain.

Both SEMs are handed to the c) simulator. Here, a tree is created. At the very beginning of the planning stage, it has only one branch, the root, which is the input of the current scene. In a trivial case the current scene already allows for the target action, which means all preconditions of the goal state are fulfilled by the current SEM. In this case "Success" is reported and a plan is handed to the robot - which consists of doing nothing except the requested action. If not all preconditions are fulfilled, it is checked in d/lea) which actions are allowed given the current scene. All allowed actions are appended to the tree as new leaves. In e) it is analyzed whether these branches contain loops. If so, they are

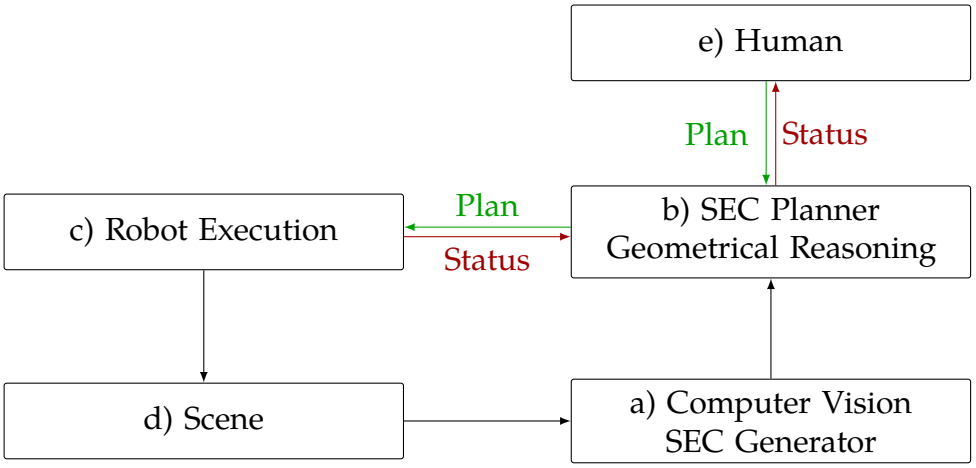


Figure 3.11.: Action perception loop of the presented system. First, d) the scene is recorded by a) a computer vision system: Here object segmentation, recognition, tracking, and eventually SEC extraction takes place. The Semantic Event Chain, as well as a labeled Point Cloud, is given to the b) SEC planner. The planner creates a plan based on a goal provided by e) a human being. The plan is given to c) a robot, which in turn will try to execute it. When encountering an error, e.g. the touching relations have changed in an unexpected way, an error signal is returned. The plan is recomputed or, if no plan is found, the error signal is escalated to the human.

removed. Also, branches that are too long are terminated. The maximum length is given to the planner as a user-defined threshold.

The tree is then given again to the c) simulator. The simulator begins at the root of the tree, which is the current scene. It takes the first branch and simulates the outcome. If the outcome fulfills all preconditions, this solution is presented to the user. Otherwise, all afforded actions of the simulated state are again appended to the current branch as leaves in d). This process is repeated for all branches. If no branch is left to check, because all have reached the user-defined threshold length, an error is reported.

This means, if one action cannot be achieved immediately, afforded actions from the ontology are tried. Iteratively, a tree is created until a solution is found or an exit criterion is met. Naturally, the here created tree grows large very fast,

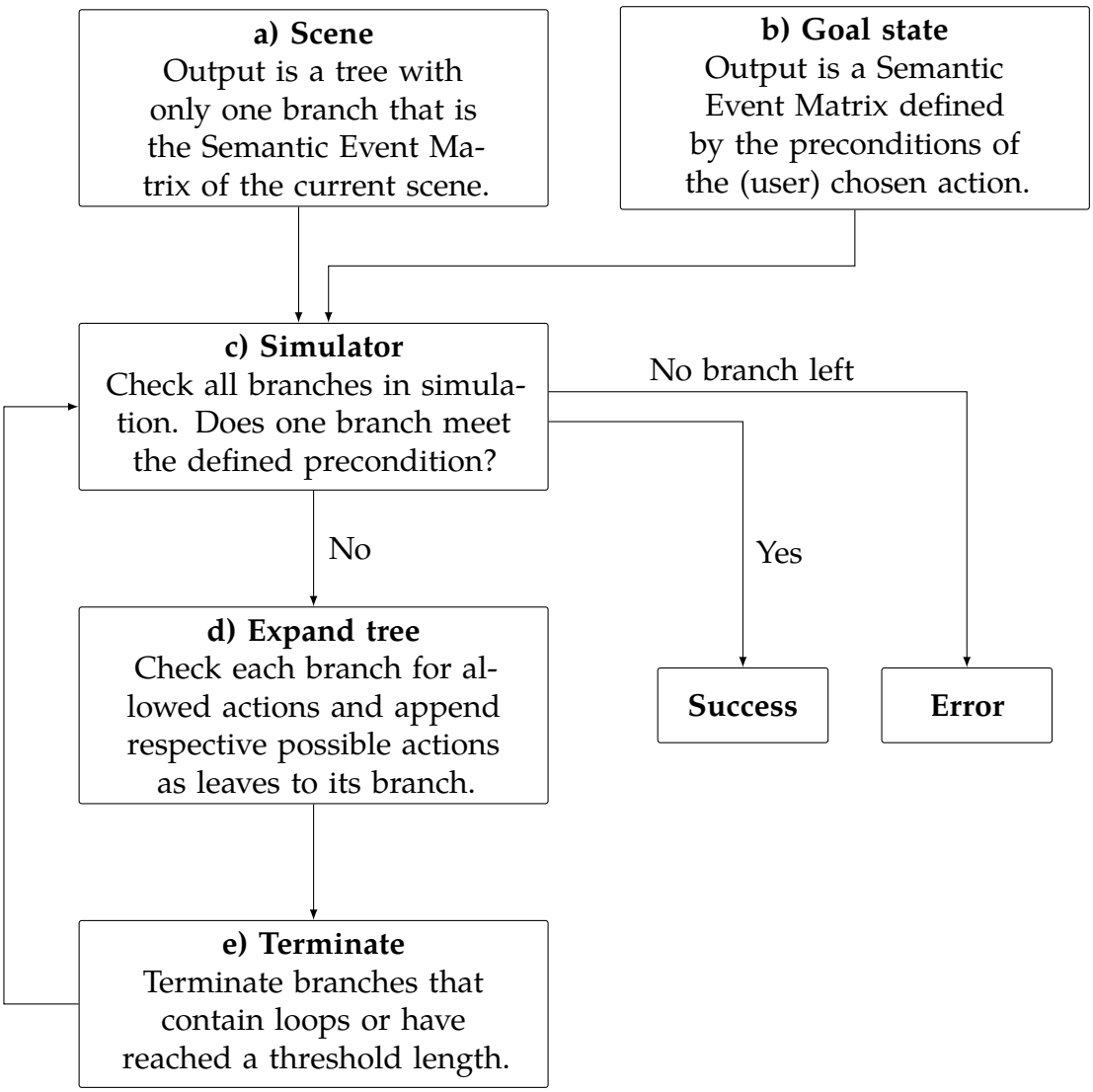


Figure 3.12.: In a) the scene is recorded and the current semantic relations are extracted. b) The goal state to the planner consists of the preconditions of the goal action that is to be performed. a) and b) are given to the c) simulator: Here, it is checked whether the preconditions of the goal state are met. If so, the plan may be executed on the robot. If no branch is left to check and no plan is found, an error message is sent. Else, the tree is expanded in d). Each branch is simulated using the postconditions from Tab. 3.4. Then, all possible actions are appended to the branch as leaves. Lastly, in e) branches that contain loops or are too long are terminated.

especially in cluttered scenes with many possible combinations of objects and actions. To minimize computational time, loop detection is implemented. The algorithm refers to loops when the same Semantic Event Matrix appears in one Semantic Event Chain more than once. Second, structural changes need to be performed to change a relation matrix, i.e. *drawing* on an object, will not make it possible to *pick and place* it later on. As suggested in [84, 130] an additional flag "changes structure of Semantic Event Chain" is attached to each action. Only if one of action contains this flag, it is added to the tree. This, again, minimizes computational complexity.

Next, the solution that is found first is not necessarily the best one and there is the possibility that several branches lead to the requested goal state. Reporting all solutions, which have a length below the user-defined threshold, calls for a measure for the best set of actions. As in Semantic Event Chains only very little structural information is stored, it is very hard to determine even the approximate trajectory length a robot has to execute. This makes it near to impossible to define a fitness function. However, one can look at the total number of actions that need to be performed and at the total number of changing touching relations, i.e. the total length of all Semantic Event Chains. This measure comes close to the execution time and allows for ranking all solutions from "least complex" to "worst" solution.

3.3. Results

3.3.1. 3d geometric reasoning algorithm

Three cluttered kitchen scenarios are used for benchmarking the algorithm; they are shown in Fig. 3.13. They are recorded using an Asus Xtion Pro camera, which records RGB as well as depth information. First, in Fig. 3.13a a cup is next to a box and an apple is on top of a pedestal. In the second scenario, Fig. 3.13b, two plates are lying around: on on top of a cutting board, the other one on a table. This plate is filled with an apple and a pedestal, and on top of the pedestal there

(a) Scene 1. **(b)** Scene 2. **(c)** Scene 3.

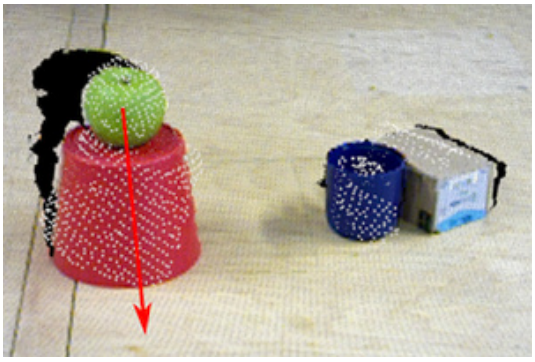
Figure 3.13.: Three different scenes are used to test the algorithm. They resemble cluttered kitchen scenarios as one might expect them in the real world.

is an empty jar. Lastly, in Fig. 3.13c, a cluttered scene containing a green apple touching a blue cup, two spoons on a spoon holder, an orange on top of a board, a knife in a knife holder, and a red pedestal in the background is introduced. While arguably all three scenarios are not very likely to be found in a kitchen outside a lab environment, they serve as a good example for cluttered objects, which are randomly stacked or lying around in close vicinity to each other.

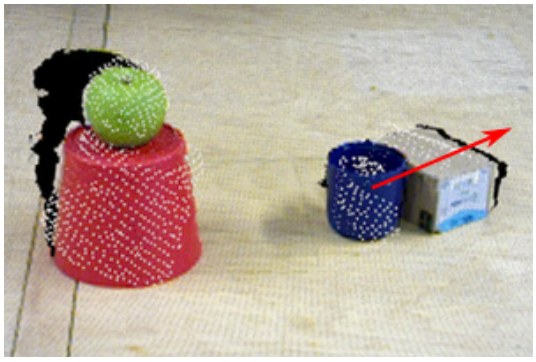
Qualitative results of geometric reasoning are shown for the first scene in Fig. 3.14, for scene 2 in Fig. 3.15, and for scene 3 in Fig. 3.16. These results show that by processing the low level point clouds one can detect the blocked and free directions of a given object. Thereafter, quantitative results are presented.

However, the quality of the direction computed also depends on the quality of the recorded point cloud data. For perfect results, points on every side of the object are needed. In real-world scenarios usually only one side is recorded, which means, that the computed normals are sometimes off. For example, in Fig. 3.15a, which shows the spatial relation between an apple and a green plate, the arrow points downwards, but also to the side. There are no points on the apple recorded, which could point downwards. The resulting arrow points to the front. However, the directions to the two objects in front of the apple, the jar (Fig. 3.15b) and yellow pedestal (Fig. 3.15c) are found correctly.

Another problem can be seen in Fig. 3.16c showing scene 3. Here, the relations between the orange spoon and the black spoon in the spoon holder (black spoon



(a) Apple and red pedestal. The arrow points to the front and bottom.

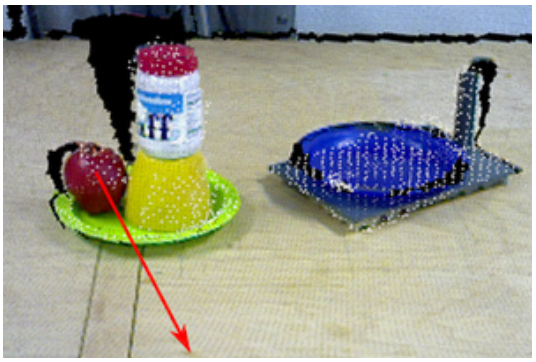


(b) Cup and box. The arrow points towards the box.

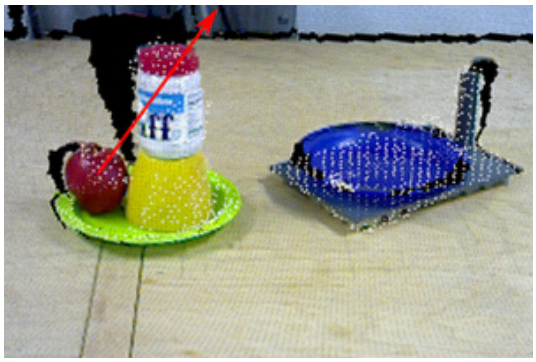
Figure 3.14.: Qualitative results for the geometrical reasoning method, scene 1. Recorded depth points on the objects are marked using white dots. The algorithm is applied to the object pair apple and red pedestal, and blue cup and box. For graphical purposes only the largest cluster is shown with a red arrow. Here, the arrow points from the apple downwards to the pedestal, which is the "forbidden" direction, if you want to lift the apple.

and spoon holder are recognized as one object) form one unexpected cluster downwards, all others point towards the spoon. Careful examinations show that there actually are some points belonging to the spoon base below the orange spoon and that the arrow downward is justified. However, the resulting access angle is very small.

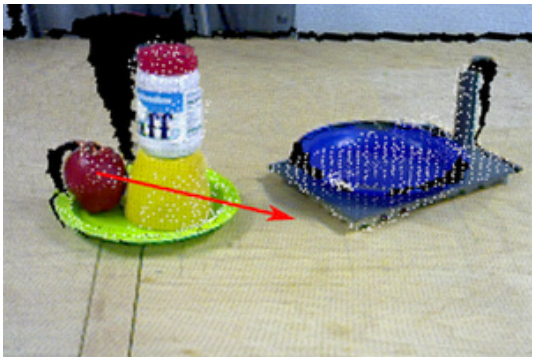
For a quantitative analysis, ground truth needs to be generated first. For the following analysis, this was done manually. For each object pair in each scene the access angles are defined. Success is reported, if and only if the 3d-cone of the found angles is entirely within the cone defined by ground truth. A negative result is issued when the cones overlap. Instead of a binary outcome, one could define a percentage of overlap as quality measure. However, false-positive errors are still dangerous in the robot execution part. Each scene is recorded ten times via an Asus Xtion Pro sensor from different angles, but always front facing — meaning the recordings are similar to the figures shown above (Fig. 3.14 – Fig. 3.16). Results are shown in Tab. 3.5, Tab. 3.6, and Tab. 3.7 for scene 1, 2, and



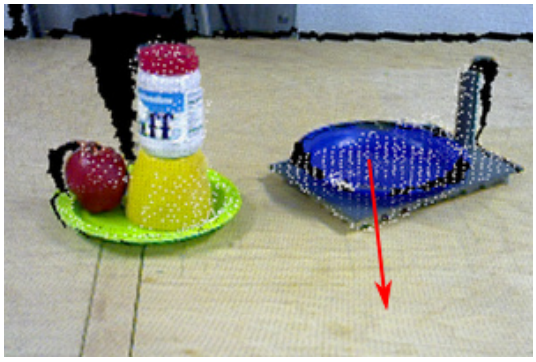
(a) Apple and green plate. The arrow is pointing to the front and not to the plate.



(b) Apple and jar. The arrow is pointing towards the jar.

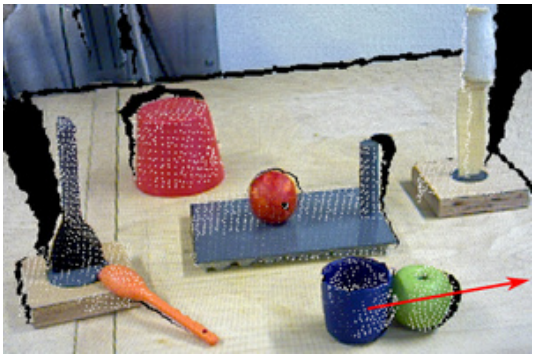


(c) Apple and yellow pedestal. The arrow is pointing towards the yellow pedestal.

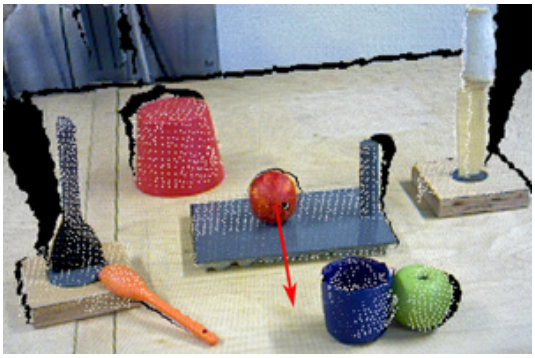


(d) Plate and board. Here, too, the arrow points as expected.

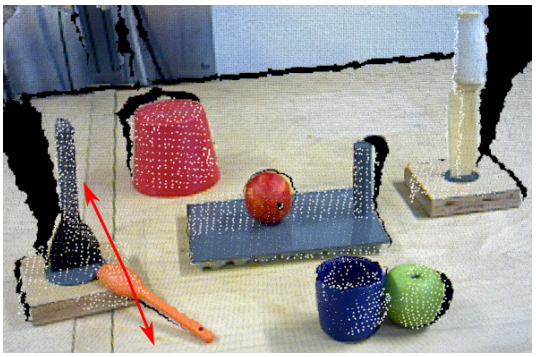
Figure 3.15.: Qualitative results for the geometrical reasoning method for scene 2. For graphical purposes only the largest cluster is shown with a red arrow.



(a) Scene 3: Blue cup and apple. The arrow points towards the apple



(b) Scene 3: Orange and board. The arrow points slightly to the front and bottom.



(c) Scene 3: Orange spoon and black spoon. The largest cluster points towards the black spoon, the second largest cluster points downwards.

Figure 3.16.: Qualitative results for the geometrical reasoning method for a cluttered scene. For graphical purposes only the largest cluster is shown with a red arrow. Please note the two red arrows in (c). Here, the two largest clusters are depicted.

	Apple	Pedestal	Blue cup	Box
Apple to		100%	100%	100%
Pedestal to	70%		70%	70%
Blue cup to	80%	80%		30%
Box to	10%	10%	10%	

Table 3.5.: Results for scene 1, see Fig. 3.13a.

	Red apple	Pedestal	Jar	Green plate	Blue plate	Cutting board
Apple to		100%	100%	0%	100%	100%
Pedestal to	20%		0%	10%	30%	30%
Jar to	100%	0%		0%	100%	100%
Green plate to	0%	0%	100%		0%	0%
Blue plate to	100%	0%	100%	0%		0%
Cutting board to	100%	100%	100%	100%	100%	

Table 3.6.: Results for scene 2, see Fig. 3.13b.

3 respectively.

They reflect the problems from the qualitative evaluation. What one first notices, is the almost binary distribution of success and failure: The percentage computed is either very high or very low. This leads to the conclusion, that under certain conditions the algorithm works well; however, border cases exist, where it fails.

First, even though the algorithm was devised to be distance independent and to offer meaningful data (even if the objects in question are far appart), the separation distance does play a role in the evaluation. Simple geometry says that objects farther away will have a smaller access cone in the ground truth data. Thus, these objects have a higher score in the above results.

	Red apple	Green apple	Blue cup	Cutting board	Pedestal
Red apple to		100%	100%	0%	0%
Green apple to	0%		20%	0%	0%
Blue cup to	100%	100%		100%	100%
Cutting board to	100%	100%	100%		30%
Pedestal to	100%	100%	100%	100%	
Knife to	-	-	-	-	-
Knife holder to	100%	100%	100%	100%	100%
Black spoon to	100%	100%	100%	100%	100%
Orange spoon to	10%	100%	100%	0%	0%
Spoon holder to	100%	100%	100%	70%	100%

	Knife	Knife holder	Black spoon	Orange spoon	Spoon holder
Red apple to	0%	0%	80%	100%	100%
Green apple to	0%	0%	0%	0%	0%
Blue cup to	100%	100%	90%	90%	90%
Cutting board to	100%	100%	100%	100%	100%
Pedestal to	100%	100%	100%	100%	100%
Knife to		-	-	-	-
Knife holder to	100%		100%	100%	100%
Black spoon to	100%	100%		100%	0%
Orange spoon to	60%	50%	0%		100%
Spoon holder to	100%	100%	100%	100%	

Table 3.7.: Results for scene 3, see Fig. 3.13c.

Next, the sensor problem needs to be discussed. As can be seen in the above visual examples, points can be only recorded on the object's side that also points to the camera. If an object is in front of another object (e.g. the red apple is in front of the red pedestal in scene 3, cf. Fig. 3.13c), but points on the front object are recorded on the "wrong" side, normals can be off. Furthermore, the points are often sparsely distributed, for example on the knife in scene 3 (cf. Fig. 3.13c) there are only very few points. The here computed normals are often erroneous, leading again to a low score.

If objects are stacked on top of each other, e.g. the jar and yellow pedestal in scene 2 (Fig. 3.13b), no points are recorded on the upper side of the bottom object. Thus, computing the direction usually fails in these cases. Since SECs store structural information about top and bottom, this does not lead to problems in the execution phase. Another problem arises, if objects are inside each other. This can be seen in scene 2 (cf. Fig. 3.13b), here the yellow pedestal is surrounded by points from the green plate. The normals of the plate point upwards, however some points of the pedestal are below the center of the plate's point cloud. By the definition of this benchmark, this leads to a failed state.

The low percentage of computed successes is a result of the strict criteria of this benchmark and the low quality of the recordings. If one relaxes the benchmark's requirements such that the center of the computed cone must be within the allowed cone of the ground truth, nearly all object pairs in all three scenes have success rates of 100%. Within this definition all cases, except the aforementioned stacked ones, can be solved correctly. For robot execution this angle is used, which allows for correct execution in all scenes.

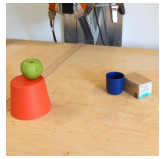
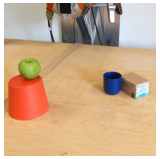
3.3.2. Scene affordance

For evaluation the same scenes as shown in Fig. 3.13 are taken. First, the effect of the top two layers of the ontology is analyzed, asking: Given a *main* object, which

actions are in principle permitted? Next, the third ontology layer performs geometric reasoning on some examples to show how actual action parametrization can be performed.

To compute the action affordances, the *main*, *primary*, and *secondary* objects are randomly selected. As there is a very high number of combinations in even very simple scenes, two different random combinations for each scene are chosen. This leads to a total of six scenarios. The results of action affordances for the three scenes are calculated by using the preconditions in the ontology and by analysis of subgraph structures. They are summarized in Tab. 3.8. Each column shows the possibility of performing different actions in the ontology for a specific selection of *main*, *primary*, and *secondary* objects. Here, one can see some limitations of the SEC domain. Some actions require additional high level object knowledge (e.g. stirring or levering) and are marked with "n"; for example stirring is always denied as it requires a liquid and a container shaped object (non-permanent objects pose a big problem for SECs or planning in general). These properties cannot be measured in the SEC domain. One could argue that also cutting, kneading, or scooping needs additional high level object knowledge, but on the touching relations level these preconditions can be ensured.

In this section it is shown that one can perform fast and computational inexpensive checks, whether a graph structure allows for execution of a specific action. In the next section, this framework is extended to also include postconditions of an action. The postconditions determine how an action changes a scene, which will enable a planning algorithm.

		Scene 1	Scene 1	Scene 2	Scene 2	Scene 3	Scene 3
							
Main object		Apple	Cup	Apple	Yellow pedestal	Orange	Apple
Primary object		Red pedestal	Box	Yellow pedestal	Green plate	Board	Cup
Secondary object		Box	Red pedestal	Blue plate	Blue plate	Cup	Board
1	punch	✓	✓	✓	✓	✓	✓
2	flick	✓	✓	✓	✓	✓	✓
3	poke	✓	✓	✓	✓	✓	✓
4	chop	✓	-	-	-	✓	-
5	bore	✓	✓	✓	✓	✓	✓
6	cut	✓	-	-	-	✓	-
7	scratch	✓	✓	✓	✓	✓	✓
8	scissor-cut	✓	-	-	-	✓	-
9	squash	✓	✓	✓	-	✓	✓
10	draw	✓	✓	✓	✓	✓	✓
11	push	✓	✓	✓	-	✓	✓
12	stir	n	n	n	n	n	n
13	knead	✓	✓	✓	-	✓	✓
14	rub	✓	✓	✓	✓	✓	✓
15	lever	n	n	n	n	n	n
16	scoop	✓	✓	✓	-	✓	✓

17	take down	✓	-	-	-	✓	-
18	push down	✓	-	-	-	✓	-
19	rip off	✓	-	-	-	✓	-
20	break off	n	n	n	n	n	n
21	uncover by pick&place	n	n	n	n	n	n
22	uncover by pushing	n	n	n	n	n	n
23	put on top	✓	-	✓	-	✓	-
24	push on top	-	-	-	-	-	-
25	put over	n	n	n	n	n	n
26	push over	n	n	n	n	n	n
27	grasp	✓	✓	✓	-	✓	✓
28	push apart	-	✓	✓	-	-	✓
29	push together	-	-	-	-	-	-
30	grasp	✓	✓	✓	-	✓	✓

Table 3.8.: The different scenes are enlarged in Fig. 3.13. Please note that one cannot check the preconditions for some actions, e.g. stirring, knead which are related to the material of objects. These actions are denoted with "n"; they require high level object knowledge. A "✓" denotes executability of the action; the actions "-" were correctly computed as not possible to execute.

3.3.3. Using affordance for planning

In this section the system is evaluated using four different scenarios. The experimental setup is as follows:

1. **Push object 1 to object 2.** In the first scenario the task is to push the *main* object, a red apple, to the *secondary* object, a yellow pedestal. Another object is touching the *secondary* object.
2. **Pick and place from tower structure.** The second scenario contains a tower structure as shown in Fig. 3.9c. The robot needs to pick and place a cutting board with fruits and needs to empty the board first.
3. **Pour liquid.** The robot needs to pour a liquid into a bowl. Currently, the bowl is used for fruits and needs to be emptied first. For the liquid colored sand is used[1].
4. **Cut cucumber.** The robot has to cut a cucumber on top of a board. The board is occupied by an apple and needs to be cleaned first.

Qualitative results for execution are shown in Fig. 3.17, Fig. 3.18, Fig. 3.19, and Fig. 3.21; corresponding to scenarios 1, 2, 3, and 4. In the next sections, we will review and discuss the findings.

Results for scenario 1

The task given to the robot system is *push the red apple to the pedestal*. The first keyframe of the scene is shown in Fig. 3.17a, the respective graph relations are as follows

[1]Experiments have shown that using real liquids for robot experiments lead to messy problems.

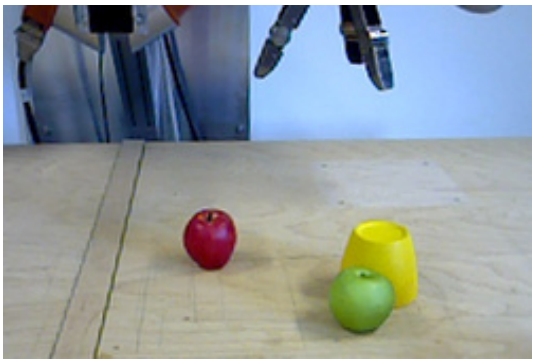
	Table	Green apple	Pedestal	Red apple
Table (id = 0)	0	T	T	T
Green apple (id = 1)	T	1	T	N
Pedestal (id = 2)	T	T	2	N
Red apple (id = 3)	T	N	N	3

The *main* object is only contained in one subgraph: one single object on top of the support (cf. Fig. 3.9a), which is, as shown in Tab. 3.3, allowed for the action *push*. Also, the preconditions for the *secondary* object are met. This is the trivial case, where the current scene already affords the preconditions of the target action. This means, that no further planning is required. After executing the action, the resulting subgraph will consist of one structure containing two touching objects on top of one support (cf. Fig. 3.9b). Keyframes showing the robot performing the requested action are listed in Fig. 3.17.

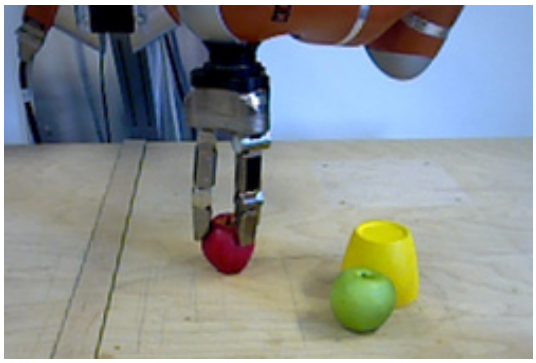
Results for scenario 2

Fig. 3.18 depicts the second scenario. The task given to the planner is to *pick and place the cutting board on top of the plate*. From the first keyframe the following graph is extracted:

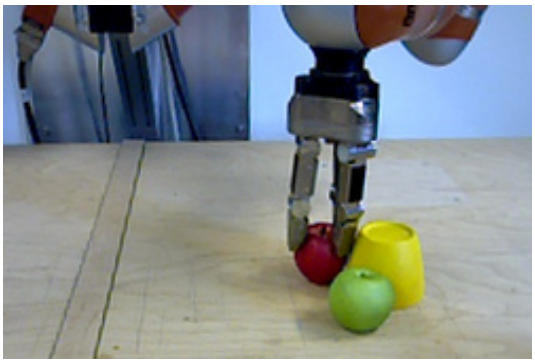
	Table	R. apple	G. apple	C. board	Plate
Table (id = 0)	0	N	N	T	T
Red apple (id = 1)	N	1	N	T	N
Green apple (id = 2)	N	N	2	T	N
Cutting board (id = 3)	T	T	T	3	N
Plate (id = 4)	T	N	N	N	4



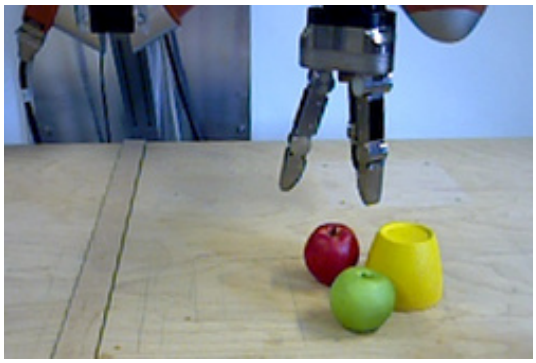
(a) Keyframe 1.



(b) Keyframe 2.



(c) Keyframe 3.



(d) Keyframe 4.

Figure 3.17.: Scenario 1: The red apple is being pushed to the pedestal, which is touched by another apple.

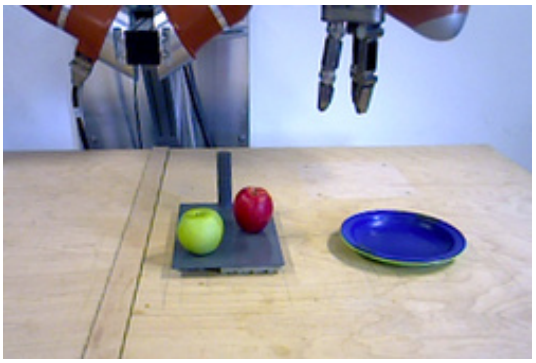
where "R. apple" is short for "Red apple", "G. apple" for "Green apple", and "C. board" stands for "Cutting board". On the semantic level the planner extracts three subgraphs around the *main* object: two are tower structures (cf. Fig. 3.9c) and not allowed for the pick and place action. The third one is one single object on top of the support, which is allowed for *pick and place*. First, the planner suggests to remove the red apple via pick and place onto the table. As the requested action is still not allowed, as one tower structure remains, a second pick and place action of the green apple is added to the list. Eventually, the cutting board is free of any other objects and can be picked and placed on top of the plate. The removal sequence of the fruits depends on the object enumeration; in the shown case the red apple is removed first as it has a lower object identifier. Successful execution of the plan

1. *Pick and place red apple on top of table*,
2. *Pick and place green apple on top of table*, and
3. *Pick and place cutting board on top of plate*.

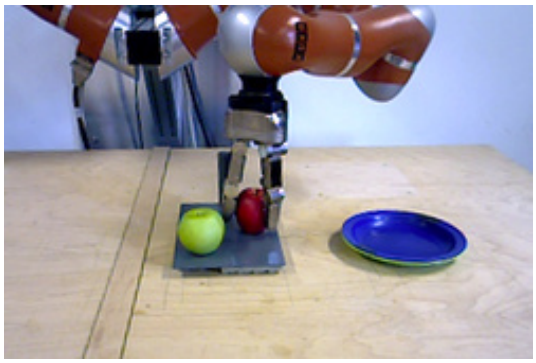
is demonstrated in Fig. 3.18.

Results for scenario 3

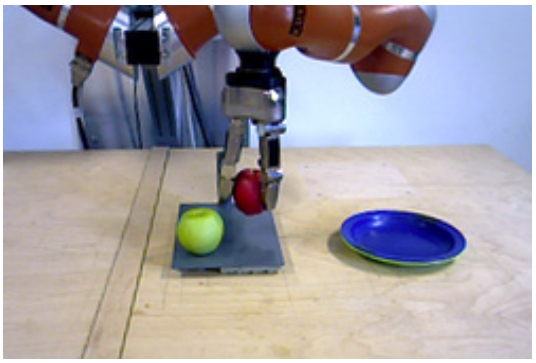
Scenario 3 holds five objects: a table (id $= 0$) acts as support, a bowl (id $= 1$), an apple (id $= 2$), and a bottle (id $= 3$). The fifth object, sauce (id $= 4$), is at the beginning hidden inside the bottle and not visible to the computer vision system. The extracted graph structure as created from keyframe Fig. 3.19a does therefore not contain sauce at the beginning. It is rather added as soon as it becomes visible and marked "absent" (A) in prior graphs of the SEC:



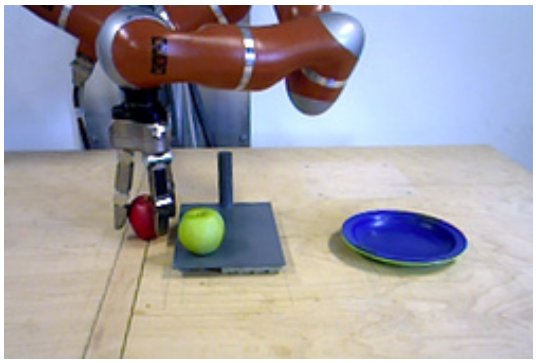
(a) Keyframe 1.



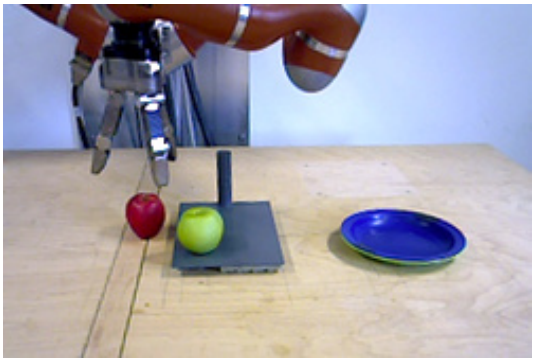
(b) Keyframe 2.



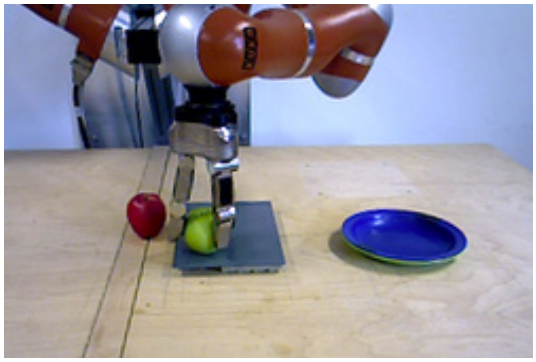
(c) Keyframe 3.



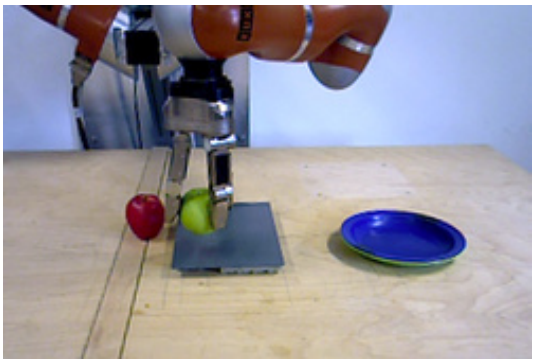
(d) Keyframe 4.



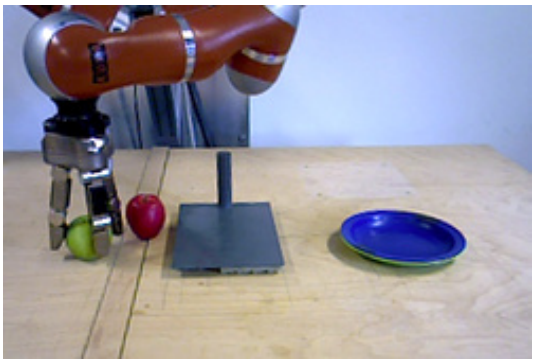
(e) Keyframe 5.



(f) Keyframe 6.

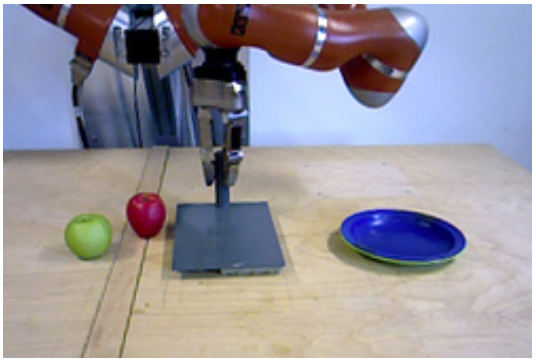


(g) Keyframe 7.

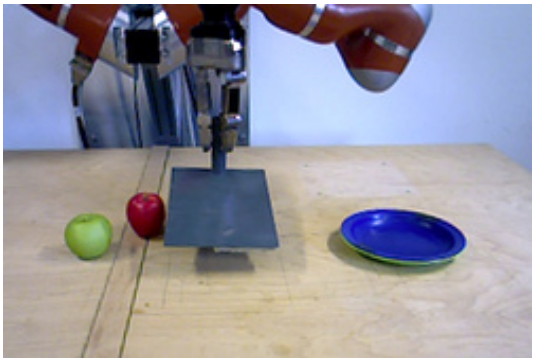


(h) Keyframe 8.

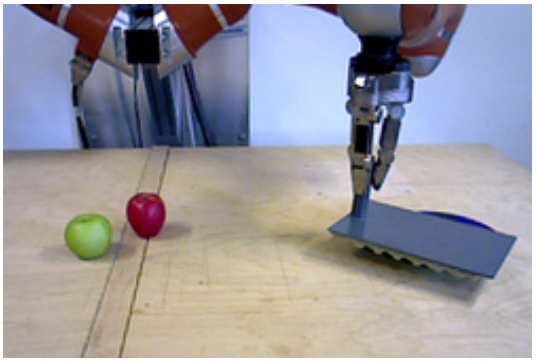
(i) Keyframe 9.



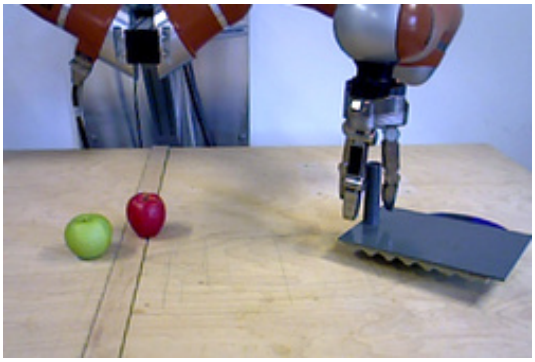
(j) Keyframe 10.



(k) Keyframe 11.



(l) Keyframe 13.



(m) Keyframe 14.

Figure 3.18.: Scenario 2: The robot needs to put a cutting board on top of another plate. For this it needs to empty the board first.

	Table	Bowl	Red apple	Bottle	Sauce
Table (id = 0)	0	T	N	T	A
Bowl (id = 1)	T	1	T	N	A
Red apple (id = 2)	N	T	2	N	A
Bottle (id = 3)	T	N	N	3	A
Sauce (id = 4)	A	A	A	A	4

Furthermore, the planning system does not contain high level knowledge about the function of objects; meaning awareness that objects may act as *containers*, which hold a liquid. As the connection between the action *pour* and an object acting as *container* for the liquid is not present, the command *pour the sauce into the bowl* will always fail due to absence of the sauce (which is not an allowed structure). While a high level planner, for example a human being, would of course first look into the bottle for sauce to pour, the command given to the planning system is changed accordingly to *pour the bottle into the bowl*. On the planning domain this is now connected to an action, where not the bottle, but rather the content of the bottle is being poured into the bowl.

The results for the third scenario are detailed in Fig. 3.19. The task is to *pour liquid from bottle into bowl*. The *primary* object, the bottle, has only one subgraph: One single object on top of support, which is an allowed structure for *pick and place*. The *main* object, however, the bowl is obstructed with an apple. Thus, the planner detects a tower structure with the bowl as forbidden element. The planner suggests to remove the top element, the apple, to afford the requested action. After the apple is removed, the sauce is being poured into the bowl. The computed plan is as follows:

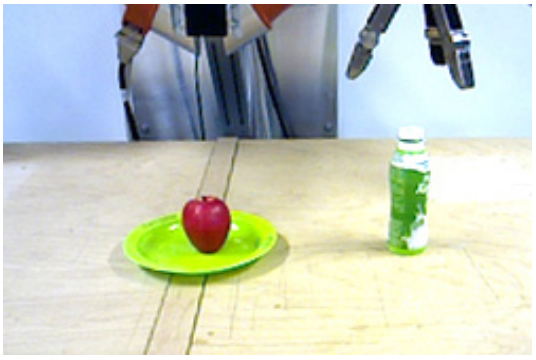
1. *Pick and place red apple on top of table*,
2. *Pour bottle into bowl*.

Results for scenario 4

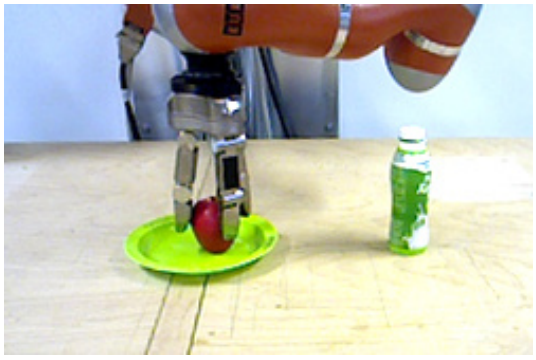
The fourth scenario can be seen in Fig. 3.21. The task is to *cut the cucumber with the knife*. The extracted graph relations are as follows:

	Table	Cucumber	G. apple	C. board	Holder	Knive
Table	0	T	N	T	T	N
Cucumber	T	1	N	N	N	N
G. apple	N	N	2	T	N	N
C. board	T	N	T	3	N	N
Holder	T	N	N	N	4	T
Knive	N	N	N	N	T	5

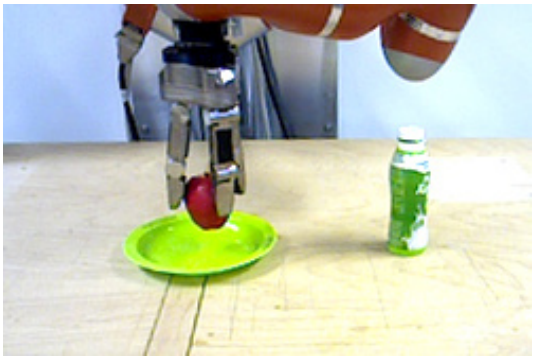
with the graph relations as shown in Fig. 3.20. Executing the action *cut the cucumber with the knife* results in immediate success. Both objects, cucumber and knife, are the only objects on top of a support. High level knowledge, i.e. "a cutting board is needed for that" is not included on this low level. Here, the human



(a) Keyframe 1.



(b) Keyframe 2.

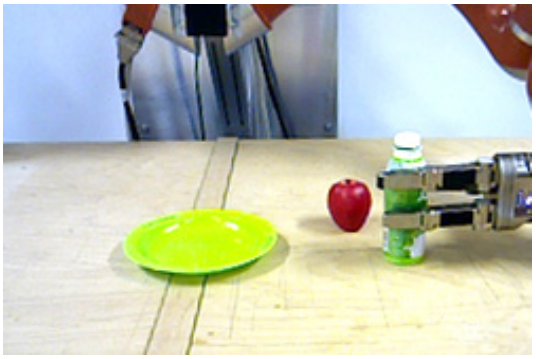


(c) Keyframe 3.

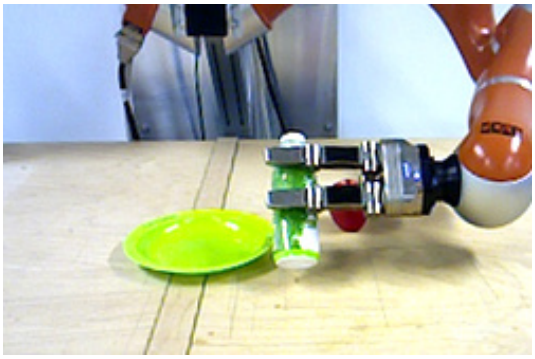
(d) Keyframe 4.

(e) Keyframe 5.

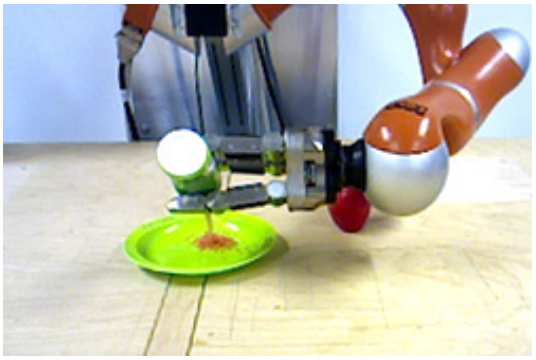
(f) Keyframe 6.



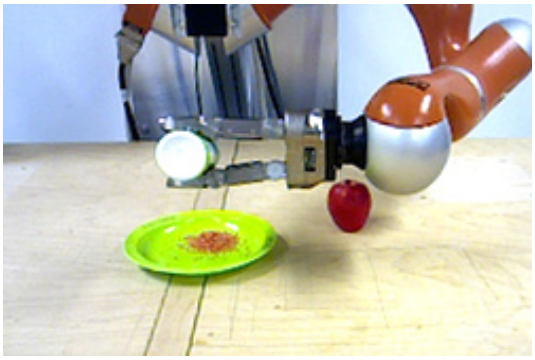
(g) Keyframe 7.



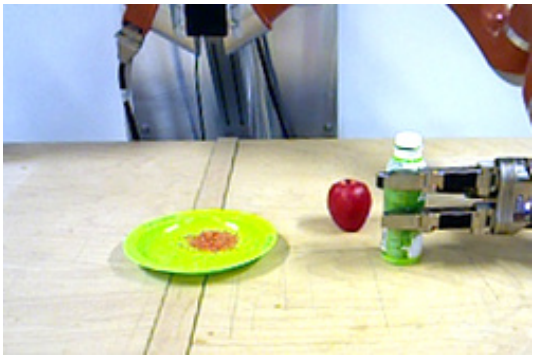
(h) Keyframe 8.



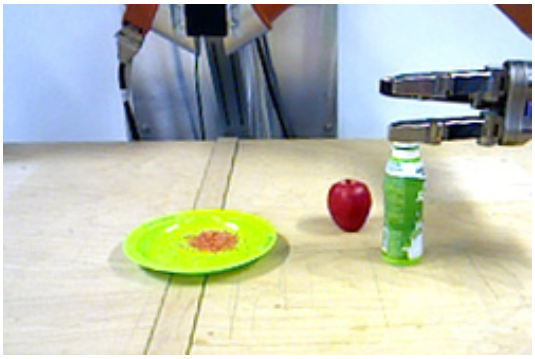
(i) Keyframe 9.



(j) Keyframe 10.



(k) Keyframe 11.



(l) Keyframe 12.

Figure 3.19.: Scenario 3: The robot needs to *pour liquid into a bowl*. Currently, the bowl is used for fruits and needs to be cleaned first.

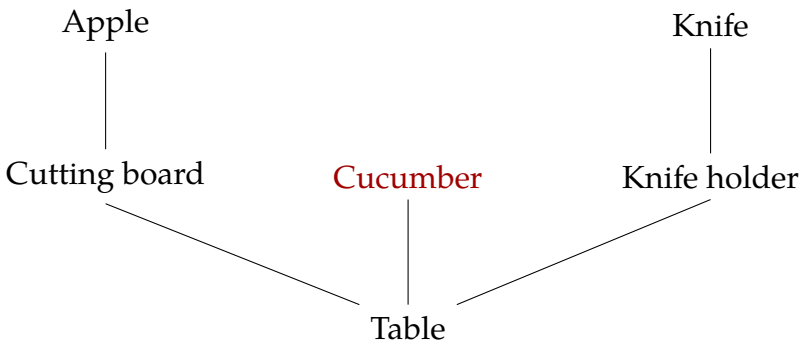


Figure 3.20.: Graph relation of the first keyframe as shown in Fig. 3.21a. The *main* object "Cucumber" is marked in red.

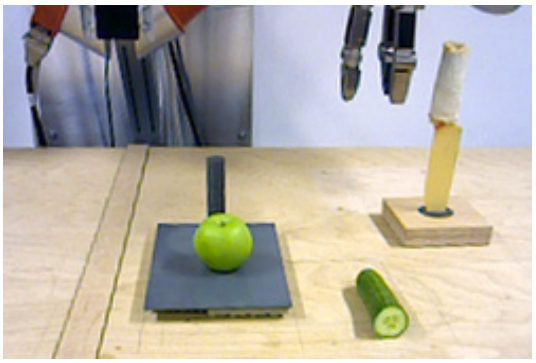
has to interfere and include the action *pick and place the cucumber on the cutting board* before *cutting*. This results in

1. *Pick and place the apple on top of the table*,
2. *Pick and place the cucumber on top of the cutting board*, and
3. *Cut the cucumber with the knife*.

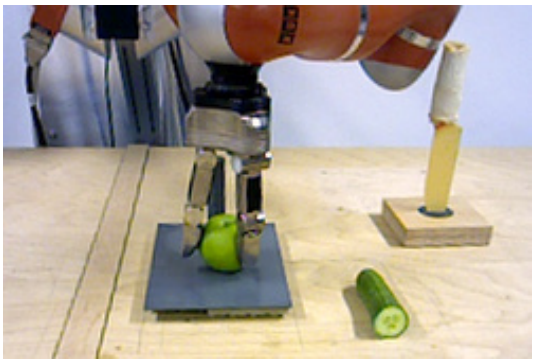
But it turns out that there is another, more practical problem. As shown in Fig. 3.21 the robot executes the above actions seemingly without any problems. The preconditions determined for *cutting* request that after cutting the *main* object must have a second object right next to it, touching it. This should be the new object coming into existence and which was cut from the cucumber. Even for human beings it is hard to say whether the last two keyframes, Fig. 3.21o and Fig. 3.21p, contain one or two pieces of cucumber. The computer vision algorithms fails, too. An error is propagated to the planning algorithm and subsequently the plan is recomputed as:

1. *Cut the cucumber with the knife*.

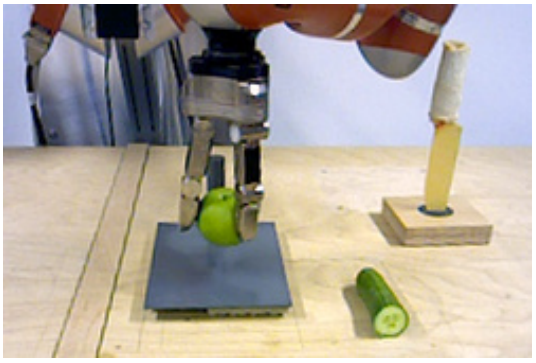
While moving the knife into cutting position, it fell out of the robot's hand and the robot was not able to grasp it again. Thus, the experiment was aborted by the user.



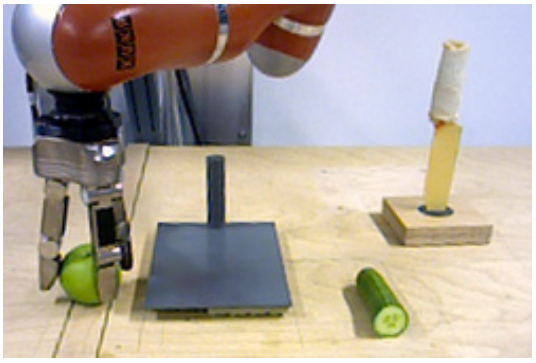
(a) Keyframe 1.



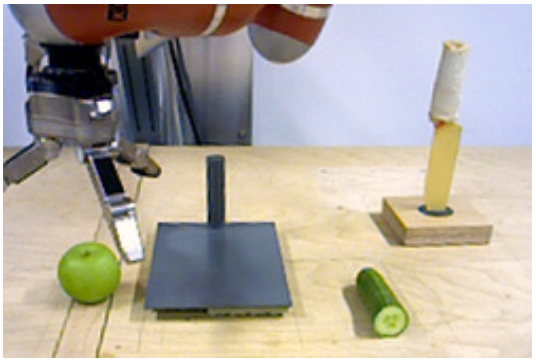
(b) Keyframe 2.



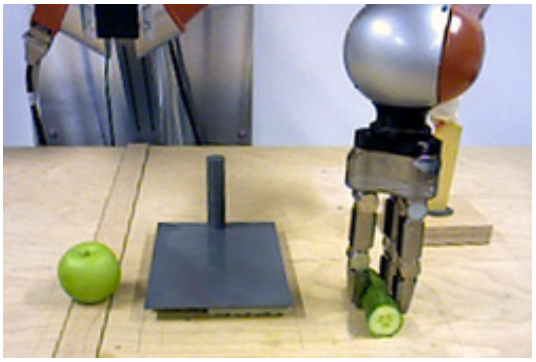
(c) Keyframe 3.



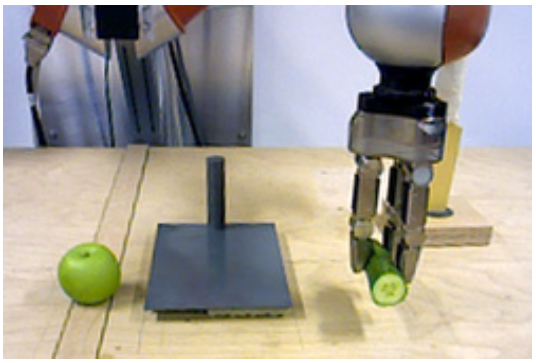
(d) Keyframe 4.



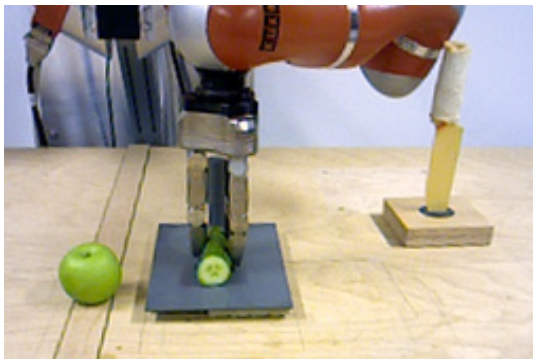
(e) Keyframe 5.



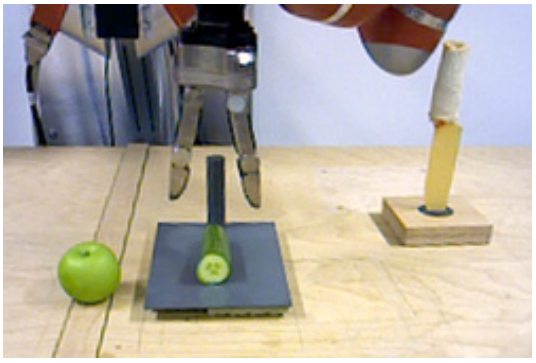
(f) Keyframe 6.



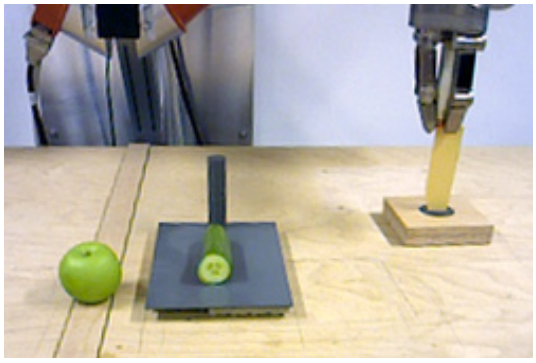
(g) Keyframe 7.



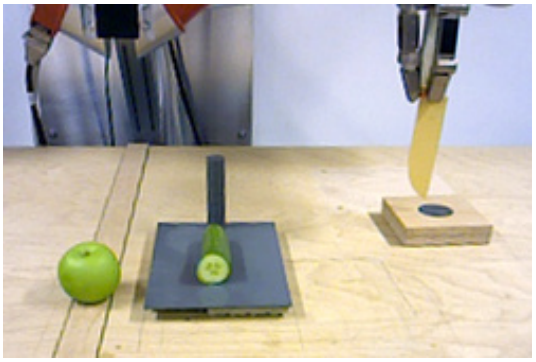
(h) Keyframe 8.



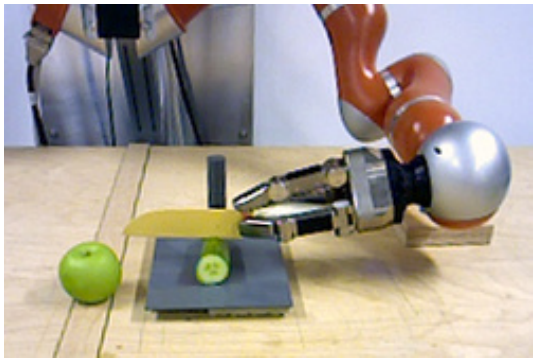
(i) Keyframe 9.



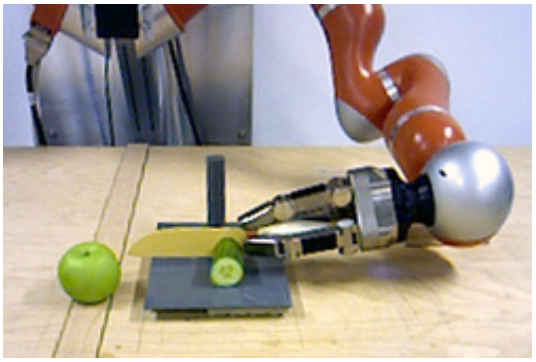
(j) Keyframe 10.



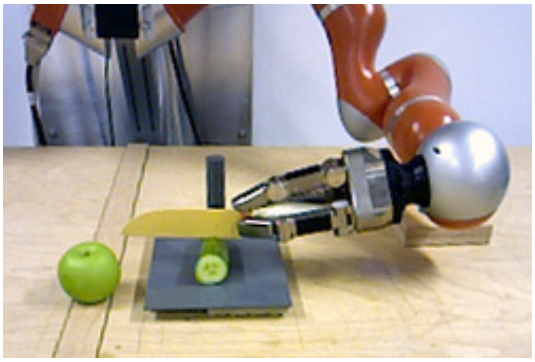
(k) Keyframe 11.



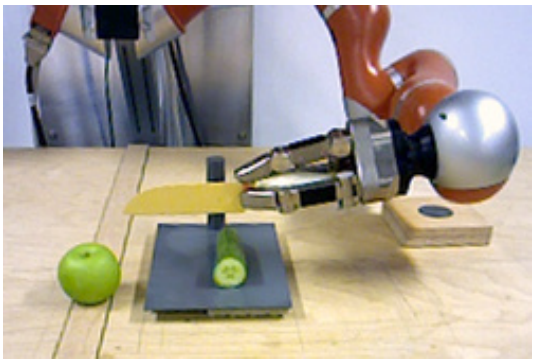
(l) Keyframe 12.



(m) Keyframe 13.

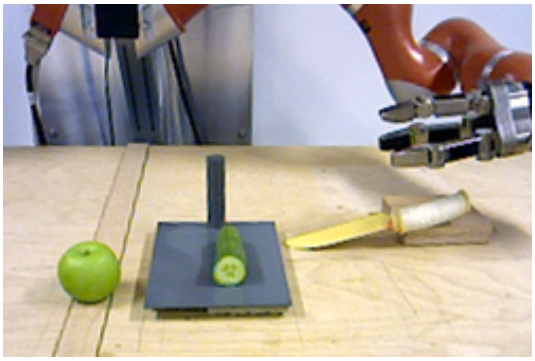


(n) Keyframe 14.



(o) Keyframe 15.

(p) Keyframe 16.



(q) Keyframe 17.

Figure 3.21.: Scenario 4: The robot needs to cut a cucumber. At the beginning the cutting board is occupied by an apple, which must be removed first.

This outlines numerous problems. First, the action *cut the cucumber with the knife* was executed incorrectly; meaning the cucumber was to be cut on the table instead of the cutting board. It becomes clear that some actions require high level knowledge for correct execution. From the list in Tab. 3.1 this includes *chop*, *cut*, *draw*, *stir*, *lever*, and *scoop*:

- *Chop*: Chopping should be performed on top of a stable support. This is the same problem which is mentioned above.
- *Cut*: See *Chop*.
- *Draw*: One can argue about *drawing* that it needs a certain amount of creativity or at least information on about what to draw. However, trajectory planning is also required for all other actions. While these trajectories usually can be generated from low level information (e.g. move the arm from point A to B and avoid obstacles while doing this), drawing needs more detailed instructions. The same could be said about *poke*, *knead*, or *rub*.
- *Stir*: The SEC domain contains no information about object details, for example as viscosity. Consequently, there is no information whether an object might be stirred.
- *Lever*: The intent of *levering* is moving a heavy object. Again, however, no detailed object information is available (as "Is this object too heavy to lift?"). Moreover, levering needs very precise information on how to use the lever: this information is not always available to the robot.
- *Scoop*: See *Stir*.

In conclusion of the first error one could add a symbolic planner on top of the SEC planner. This is discussed in the next session.

Second, there is the sensor problem that the cucumber was not detected as being successfully cut. The here utilized experiment setup is vision based. One could add other sensors provided by the robot: for example force or the mere fact that the knife has cut through the cucumber and touched the board. Especially the

second fact can often be detected on the relation graphs and the generation of keyframe Fig. 3.21m shows that it indeed was detected here. When to trust vision and when to trust relation graphs (which also are generated from vision here!) is another question. As neither has a score measuring uncertainty, this remains an open question.

Lastly, there is the hardware problem that the robot drops the knife. The cutting movement rocked the knife such that the robot's grip became unstable. Even though the robot's hand contains force feedback sensors, this was not detected. After the knife fell down, the robot is not able to grasp it again in a secure manner or rotate the knife such that the sharp edge points downwards. However, what one could do in such an event is to define an error routine:

1. *Pick and place the knife into the knife holder*,
2. *Cut the cucumber with the knife*,

where the knife is first put in a well defined position where it can be grabbed again in secure manner. This error routine is hardware- and action-dependent and does not generalize very well. Again, it holds higher level knowledge about specific outcomes of specific events.

3.4. Discussion

3.4.1. 3d geometric reasoning algorithm

Semantic Event Chains can be created very efficiently from a 6d movie stream. Computing the relative pose between two objects is, however, computationally very expensive and thus should not be computed for every frame. It only needs to be recomputed, if an object pose changes either by active robot movement or by external force.

It is shown that the quality of the results relies on the quality of the recorded point cloud data. The camera output depends on many different conditions, which are already discussed above in Sec. 2.3.7. Enforcing a strict definition such that the computed cone of allowed access angles must always be free of the object fails in many real-world scenarios. Here, the recorded points of an object often are on the "wrong" side. Furthermore, smaller objects may have not enough points to compute normals in a stable manner.

Relaxing these conditions such that only the center of the computed cone must be free of objects leads to an almost 100% success rate. Only stacked objects fail in this case. Stacked objects do not pose a problem in the execution phase as this information is also encoded in SECs.

3.4.2. Scene affordance

In this experiment, structural information is taken into consideration and solves many problems. However, here two new issues surface: First, physical properties play an important role in action execution. Arguably, even though on the SEC level, cutting an apple with a pedestal (cf. Tab. 3.8) is feasible, this certainly is not possible in a real world experiment. The same is true for many actions, which change the appearance of an object (i.e. all destroy actions like cutting, breaking; but also drawing or scooping). Here, high level symbolic knowledge needs to be included, which offers information on tool usage and object properties. Second, no objects which are skewed or slanted are included. This means, objects can in most cases be stacked and do not move due to the support's slope. Both issues stem from the reduced view of physical properties in the SEC domain. Trying to overcome these problems results in an almost complete physical simulation and the advantage of Semantic Event Chains would be lost. Instead, error handling for robot execution is introduced. This is discussed in the next section.

3.4.3. Using affordance for planning

One of the main goals of this work is to find a planning mechanism, which does not rely on high level knowledge, is hardware-agnostic, and yet remains powerful enough to solve basic tasks. The Semantic Event Chains provided by Aksoy et al. [4] are based on hardware-independent relation changes; path and motion planning is performed on the abstraction layer below the SECs as they are robot-dependent. The domain used is provided by Wörgötter et al. [133], even though it was found that the ontology used is not complete, e.g. *push object A* is treated differently than *push object A and B together* or *push object A and B apart*.

The comparison to contemporary high level planning systems becomes difficult. One can always define a symbol in those planners as one touching relation with defined pre- and post conditions. In such a case an action execution would only reflect the transfer from one relation to another. Therefore, each action can be implemented in such a planner and solved with contemporary domain definition languages (e.g. STRIPS [43] or PDDL [77]). However, the goal was to find an additional low level planning layer, which solves problems without high level knowledge but with basic structural knowledge. It operates on a limited domain, which imposes several constraints. Yet, the shown action domain proves to be powerful enough for almost all every day tasks. It is shown that only for very specific tasks high level knowledge is required, which reduces the complexity of the high level planning algorithm significantly.

One limitation of this approach is the fixed repository of pre- and postconditions. In [5] it was shown that action bootstrapping based on SECs is possible. Currently, it is investigated how affordances and effects of an action can be learned. This might happen in an environment, where a robot randomly assesses different configurations and actions on a given set of objects. In human development this is, for example, known to happen when a toddler plays in an undirected manner with building blocks.

Another limitation is mentioned in Sec. 3.3.3, where high level knowledge is important for correct task execution. Instead of integrating this knowledge into

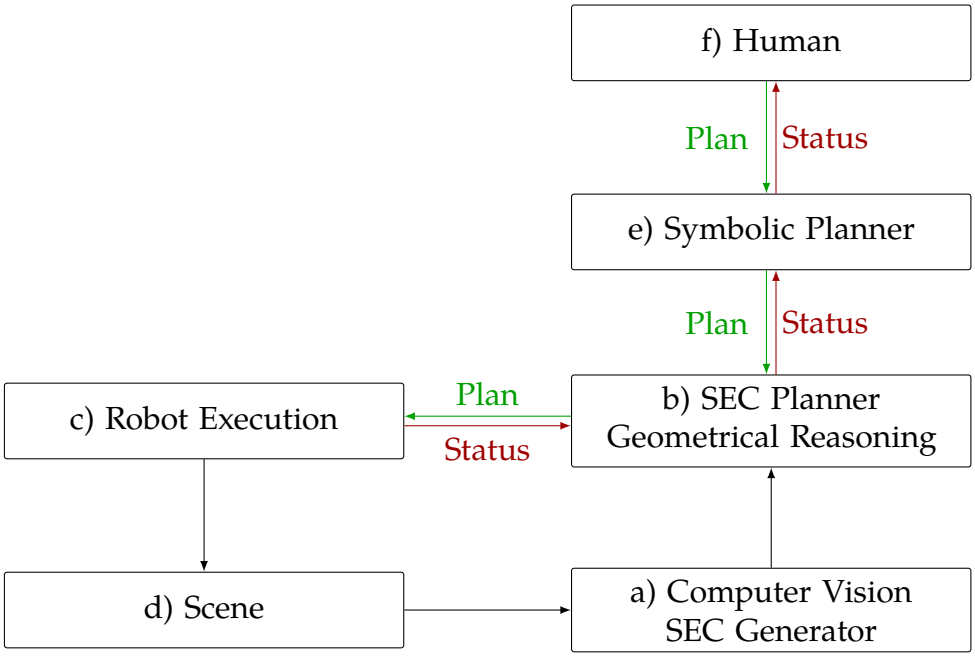


Figure 3.22.: The structure resembles Fig. 3.11, but includes e) a symbolic planner. This planner includes high level knowledge as object properties or functions of objects.

the low level planner (and therefore loose the ability to generalize on almost any arbitrary action sequence and combination), it can be outsourced to a high level planner. This is displayed in Fig. 3.22 which resembles Fig. 3.11, but a symbolic planner is added between the SEC planner and human. In this work, a planner as described in [2] is used. The human being enters the plan to the symbolic planner, which in turn analyzes it based on a predefined domain: For example, "a cutting action must always take place on top of a cutting board" or "only liquids may be stirred". The symbolic planner expands the user-defined goal action to a list of atomic actions, which are handled by the SEC planner. Problems listed above are circumvented using this method, however, only those issues may be addressed, which are already included in the symbolic planner. Here, generalization is lost, even though the planning mechanism allows for learning these preconditions by trial and error. The advantage of this method is that always the lowest possible layer of planning is used. This reduces the level of complexity in the high level planner significantly.

The here shown planning approach is entirely data-driven and bottom up. This means, it generalizes well, except for those actions, which require high level knowledge: Here a second planner is needed. This enables robots to plan complex action sequences in unknown — and possibly even unstructured — environments. Apart from knowledge about tools, e.g. a knife for cutting, the approach is even agnostic of object functions, size, or shape. The resulting planning system bridges the signal-to-symbol gap in a natural way and allows for rapid planning even in complex environments.

4

Conclusion and outlook

In this work, two robot systems are analyzed in detail. Both perform as closed-loop systems in the robotic action-perception loop. The first system consists of three robots: two wheeled and one flying agent, which all three share the same hardware base. Here, the focus lies on the perception side of the loop. The second system assesses the question of low level planning and action domain. The action side of the loop is discussed in great detail in this part.

For the first system a low level denoising algorithm is introduced, which is able to filter RGB images but also generalizes on any dimension. Because of this property the filter is used on 1d data on the robots. It is shown that it outperforms other local methods in quality. Global methods produce better results, but are computational to expensive to run on embedded hardware. In a second part of this chapter, the perception side is analyzed in further detail. The robots are equipped with omnidirectional cameras, which allow for stable feature detection even when the robot changes poses rapidly. On simulation it is shown that the developed algorithms outperform current state-of-the-art. The results from this chapter enable truly autonomous flying robots in indoor, GPS-denied environments. This includes, but is not limited to, indoor building inspections after, for example, earthquakes, underground search-and-rescue in mining disasters, and systems, which need to have a fall-back setting if the GPS system fails.

Some questions still remain open and should be studied further. First, the quality of the filter on 2d can still be enhanced greatly. In all methods, a noisy pixel is detected based on some threshold. Global methods, currently most prominently deep learning algorithms, search for similar patches in the entire image and try

to replace the noisy area. This is, however, computationally very expensive. Local methods replace noisy pixels based on a predefined local neighborhood. One could extend the here presented sliding window approach to search for similar patches around a noisy pixel in a local neighborhood. This combines the quality deep learning methods bring along with the speed of local methods.

There are many follow-up ideas for the quadcopter project, too:

- Evaluation of long term flights via tracking the robot's position with external cameras. For example, high precision Vicon cameras offer sub millimeter accuracy at 120 Hz [131].
- Currently, at each frame a full pose update is computed. This is, however, computationally expensive. One could, based on the last pose updates and based upon the assumption that on small time scales not too much changes, compute a new pose via VO only, if the IMU indicates a large change.
- This method is fully data driven. This bottom-up method has proven to be useful in an unknown and possibly unstructured environment, e.g. inside a house after an earthquake. However, one could use some high level features, e.g. "door frame might be an opening to adjacent room" for navigation.
- As currently a map is built, there is no loop closure detection. This would lead to a full Simultaneous Localization and Mapping approach.
- Last, one should devise more benchmarks for evaluation against other methods. However, creating a fully simulated environment is very tedious.

Second, the VO's depth acquisition algorithm needs to be analyzed in more detail. This means, the computed depth of points needs to be compared to ground truth information, preferable in a real-world measurement and not in simulation. One feature's error depends on multiple settings: most prominently the camera's resolution, how stable the feature can be tracked, and it depends heavily on the error of all other found features (as the pose update is computed from

all features). How to find a good error measure is currently ongoing research. An understanding of the error margins can be used to track each feature using an EKF. Permanent tracking allows for non-static features. Furthermore, if the robot returns to a previously visited place, it could redetect the features and perform loop closure detection. In VO the error accumulates over time and loop closure would raise performance significantly.

The second system consists of a two-armed robot. The perception side, which is not analyzed in more detail here, of this loop generates abstract graph relations of the current scene called SEC. Each node in this graph holds one object, an edge is added if two objects touch. This graph is enriched with pose relations between objects, which is computed using a 3d geometric reasoning algorithm. An ontology of actions shown in [133] is used where each action is connected to a set of pre- and postconditions, which are also defined on the SEC domain. It is first shown that one can compute a scene's action affordance based on the preconditions. Next, it is shown that the postconditions allow for simulation of an action in the SEC domain. This enables complex planning, which is entirely data-driven and bottom up: Meaning unknown or unstructured environments do not pose a problem and the signal-to-symbol can be bridged in a natural occurring way. However, it is also shown that for some actions high level knowledge is required, e.g. the action *cutting a tomato with a knife* should not happen directly on a table — but instead on a cutting board. To circumvent this problem, a high level planning architecture is included, which parses human input based on predefined symbols and preconditions.

Here, too, a few open research questions remain. First, can a robot learn the set of postconditions? Based on Semantic Event Chain this means to reliably predict the changes that occur in subgraphs while performing an action. If so, the next question that arises is to also learn the preconditions of an action. The robot needs to decide when an action can be performed free of error. However, both of these ongoing research questions implicitly expect a previously known repository of actions.

In a very last step one could devise an experiment, in which a robot has nei-

ther knowledge about pre- and postconditions, but also only very limited knowledge about actions. Only random movements towards objects and grasping are known. As input to the learning algorithm ESEC only are given. Given the time, the robot would soon find out about the effect of, for example, *pick & place*, *pushing*, and letting objects *drop*. The average time to explore an action could be used as a measure for difficulty or complexity. One would expect that actions that perform structural changes can be learned faster than others, for example, scratch or draw. This could hint at the fact that some actions are learned by trial-and-error during undirected play, while others are found by imitation.

Bibliography

[1] M. J. Aein et al. "Toward a library of manipulation actions based on semantic object-action relations". In: *International Conference on Intelligent Robots and Systems*. IEEE/RSJ. Nov. 2013, pp. 4555–4562.

[2] A. Agostini, C. Torras, and F. Wörgötter. "Efficient interactive decision-making framework for robotic applications". *Artificial Intelligence* 247 (2017), pp. 187–212.

[3] E. E. Aksoy et al. "Execution of a Dual-Object Pushing Action with Semantic Event Chains". In: *Int. Conf. on Humanoid Robots*. IEEE-RAS. 2011, pp. 576–583.

[4] E. E. Aksoy et al. "Learning the semantics of object-action relations by observation". *The International Journal of Robotics Research* 30 (Sept. 2011), pp. 1229–1249.

[5] E. E. Aksoy et al. "Structural bootstrapping at the sensorimotor level for the fast acquisition of action knowledge for cognitive robots". In: *International Conference on Development and Learning and Epigenetic Robotics*. IEEE. Osaka (Japan), Aug. 2013, pp. 1–8.

[6] E. E. Aksoy. "Semantic analysis of image sequences using computer vision methods". PhD thesis. Göttingen: Georg-August-Universität Göttingen, Oct. 2012.

[7] E. E. Aksoy et al. "Learning the semantics of object–action relations by observation". *The International Journal of Robotics Research* 30.10 (2011), pp. 1229–1249.

[8] L. Alvarez, P.-L. Lions, and J.-M. Morel. "Image selective smoothing and edge detection by nonlinear diffusion. II". *SIAM Journal on numerical analysis* 29.3 (1992), pp. 845–866.

[9] AMD. *AMD EPYC 7601 CPU specifications*. 2018. URL: https://www.amd.com/de/products/specifications/cpu (visited on 03/27/2018).

[10] P. Arbelaez et al. "Contour detection and hierarchical image segmentation". *Transactions on pattern analysis and machine intelligence* 33.5 (2011), pp. 898–916.

[11] R. C. Arkin. "Integrating behavioral, perceptual, and world knowledge in reactive navigation". *Robotics and Autonomous Systems* 6.12 (1990). Designing Autonomous Agents, pp. 105–122.

[12] F. Augugliaro et al. "Knot-tying with flying machines for aerial construction". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. 2015, pp. 5917–5922.

[13] F. Augugliaro et al. "The flight assembled architecture installation: Cooperative construction with flying machines". *Control Systems* 34.4 (2014), pp. 46–64.

[14] B. Aune. "Action and ontology". *Philosophical Studies* 54.2 (1988), pp. 195–213.

[15] B. Aune. *Reason and Action*. Netherlands: Springer, 1977, p. 207.

[16] H. Bay, T. Tuytelaars, and L. Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[17] C. Belta et al. "Symbolic planning and control of robot motion [grand challenges of robotics]". *Robotics & Automation Magazine* 14.1 (2007), pp. 61–70.

[18] R. Bischoff et al. "The KUKA-DLR Lightweight Robot arm - a new reference platform for robotics research and manufacturing". In: *41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK)*. June 2010, pp. 1–8.

[19] J.-Y. Bouguet. "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm". *Intel Corporation* 5.1-10 (2001), p. 4.

[20] C. Breazeal and B. Scassellati. "Robots that imitate humans". *Trends in cognitive sciences* 6.11 (2002), pp. 481–487.

[21] D. Brescianini and R. D'Andrea. "Computationally Efficient Trajectory Generation for Fully Actuated Multirotor Vehicles". *Transactions on Robotics* 34.3 (June 2018), pp. 555–571.

[22] A. Buades, B. Coll, and J.-M. Morel. "A non-local algorithm for image denoising". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. IEEE. June 2005, pp. 60–65.

[23] O. Caldiran et al. "Bridging the gap between high-level reasoning and low-level control". In: *Logic Programming and Nonmonotonic Reasoning*. Springer, 2009, pp. 342–354.

[24] M. Chino et al. "Preliminary estimation of release amounts of 131I and 137Cs accidentally discharged from the Fukushima Daiichi nuclear power plant into the atmosphere". *Journal of nuclear science and technology* 48.7 (2011), pp. 1129–1134.

[25] H. M. Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[26] W. Clark, J. Golinski, and S. Schaffer. *The sciences in enlightened Europe*. University of Chicago Press, 1999.

[27] K. Dabov et al. "Image denoising by sparse 3-D transform-domain collaborative filtering". *Transactions on image processing* 16.8 (2007), pp. 2080–2095.

[28] G. De Giacomo et al. "IndiGolog: A high-level programming language for embedded reasoning agents". In: *Multi-Agent Programming:* Springer, 2009, pp. 31–72.

[29] C. Demonceaux, P. Vasseur, and C. Pegard. "Omnidirectional vision on UAV for attitude computation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. May 2006, pp. 2842–2847.

[30] J. Deng et al. "Imagenet: A large-scale hierarchical image database". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2009, pp. 248–255.

[31] M. G. Diaz et al. "Analysis and evaluation between the first and the second generation of RGB-D sensors". *Sensors Journal* 15.11 (2015), pp. 6507–6516.

[32] D. L. Donoho. "De-noising by soft-thresholding". *Transactions on information theory* 41.3 (1995), pp. 613–627.

[33] W. Du, X. Tian, and Y. Sun. "A dynamic threshold edge-preserving smoothing segmentation algorithm for anterior chamber OCT images based on modified histogram". In: *4th International Congress on Image and Signal Processing (CISP)*. Vol. 2. IEEE. 2011, pp. 1123–1126.

[34] D. Eberli et al. "Vision based position control for MAVs using one single circular landmark". *Journal of Intelligent & Robotic Systems* 61.1–4 (2011), pp. 495–512.

[35] J. Engel, J. Sturm, and D. Cremers. "Scale-Aware Navigation of a Low-Cost Quadrocopter with a Monocular Camera". *Robotics and Autonomous Systems (RAS)* 62.11 (2014), pp. 1646–1656.

[36] J. Engel, V. Koltun, and D. Cremers. "Direct sparse odometry". *Transactions on Pattern Analysis and Machine Intelligence* (2017).

[37] J. Engel, T. Schöps, and D. Cremers. "LSD-SLAM: Large-scale direct monocular SLAM". In: *European Conference on Computer Vision*. Springer. 2014, pp. 834–849.

[38] J. Engel, J. Sturm, and D. Cremers. "Accurate figure flying with a quadrocopter using onboard visual and inertial sensing". *Imu* 320 (2012), p. 240.

[39] E. Erdem et al. "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. May 2011, pp. 4575–4581.

[40] E. Erdem et al. "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 4575–4581.

[41] P. Eyerich, T. Keller, and B. Nebel. "Combining action and motion planning via semantic attachments". In: *Proceedings of ICAPS Workshop on Combining Action and Motion Planning*. AAAI Press, May 2010.

[42] D. Falanga et al. "Vision-based Autonomous Quadrotor Landing on a Moving Platform". In: *International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE/RSJ. 2017, pp. 1–8.

[43] R. E. Fikes and N. J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*. IJCAI'71. London, England: Morgan Kaufmann Publishers Inc., 1971, pp. 608–620.

[44] C. Forster, M. Pizzoli, and D. Scaramuzza. "SVO: Fast semi-direct monocular visual odometry". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. May 2014, pp. 15–22.

[45] C. Forster et al. "Svo: Semidirect visual odometry for monocular and multicamera systems". *Transactions on Robotics* 33.2 (2017), pp. 249–265.

[46] M. El-gayar, H. Soliman, and N. Meky. "A comparative study of image low level feature extraction algorithms". *Egyptian Informatics Journal* 14.2 (2013), pp. 175–181.

[47] D. L. Gera. *Ancient Greek ideas on speech, language, and civilization*. Oxford University Press, USA, 2003.

[48] B. Goertzel. "Artificial general intelligence: concept, state of the art, and future prospects". *Journal of Artificial General Intelligence* 5.1 (2014), pp. 1–48.

[49] B. Goertzel, M. Iklé, and J. Wigmore. "The architecture of human-like general intelligence". In: *Theoretical Foundations of Artificial General Intelligence*. Springer, 2012, pp. 123–144.

[50] D. Goldin, S. Smolka, and P. Wegner. *Interactive Computation: The New Paradigm*. Berlin: Springer, 2009, p. 487.

[51] R. C. Gonzalez and R. E. Woods. *Digital image processing*. Pearson Prentice Hall, 2002.

[52] S. Gu et al. "Weighted nuclear norm minimization with application to image denoising". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2014, pp. 2862–2869.

[53] H. Haggag et al. "Measuring depth accuracy in RGBD cameras". In: *7th International Conference on Signal Processing and Communication Systems (ICSPCS)*. IEEE, Dec. 2013, pp. 1–7.

[54] K. Hauser and J.-C. Latombe. "Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries". *ICAPS Workshop on Bridging the gap between task and motion planning* (2009), pp. 19–23.

[55] G. Havur et al. "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 445–452.

[56] J. Heinly, E. Dunn, and J.-M. Frahm. "Comparative Evaluation of Binary Features". In: *12th European Conference on Computer Vision (ECCV)*. Ed. by A. Fitzgibbon et al. Berlin: Springer, Oct. 2012, pp. 759–773.

[57] L. P. Kaelbling and T. Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2011, pp. 1470–1477.

[58] R. E. Kalman. "A new approach to linear filtering and prediction problems". *Journal of basic Engineering* 82.1 (1960), pp. 35–45.

[59] C. Kerl, J. Sturm, and D. Cremers. "Robust odometry estimation for RGB-D cameras". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2013, pp. 3748–3754.

[60] S. G. C. KG. *Servo-electric 3-Finger Gripping Hand SDH*. http://www.schunk-modular-robotics.com/en/home/products/servo-electric-3-finger-gripping-hand-sdh.html. 2018.

[61] Y. Kim. "Convolutional neural networks for sentence classification". *arXiv preprint arXiv:1408.5882* (2014).

[62] King James I. of England. *King James Bible*. Cambridge University Press, 1611.

[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

[64] U. Kuter et al. "Task decomposition on abstract states, for planning under nondeterminism". *Artificial Intelligence* 173.5 (2009), pp. 669–695.

[65] D. C. Lee, M. Hebert, and T. Kanade. "Geometric reasoning for single image structure recovery". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2009, pp. 2136–2143.

[66] A. Lev, S. W. Zucker, and A. Rosenfeld. "Iterative enhancemnent of noisy images". *Transactions on Systems, Man, and Cybernetics* 7.6 (1977), pp. 435–442.

[67] Y.-M. Liang et al. "Learning Atomic Human Actions Using Variable-Length Markov Models". *Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 39.1 (Feb. 2009), pp. 268–280.

[68] D. Lin et al. "Learning important spatial pooling regions for scene classification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2014, pp. 3726–3733.

[69] T.-Y. Lin et al. "Microsoft coco: Common objects in context". In: *European Conference on Computer Vision*. Springer. 2014, pp. 740–755.

[70] M. Lindenbaum, M. Fischer, and A. Bruckstein. "On Gabor's contribution to image enhancement". *Pattern Recognition* 27.1 (1994), pp. 1–8.

[71] W. LLC. *AMD EPYC 7601*. 2018. URL: https://en.wikichip.org/wiki/amd/epyc/7601 (visited on 03/27/2018).

[72] D. G. Lowe. "Object recognition from local scale-invariant features". In: *The proceedings of the seventh IEEE international conference on Computer vision*. Vol. 2. IEEE. 1999, pp. 1150–1157.

[73] R. Lukierski, S. Leutenegger, and A. J. Davison. "Room layout estimation from rapid omnidirectional exploration". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. May 2017, pp. 6315–6322.

[74] J. Mairal et al. "Non-local sparse models for image restoration". In: *12th International Conference on Computer Vision*. IEEE. 2009, pp. 2272–2279.

[75] L. F. Marin-Urias et al. "Towards shared attention through geometric reasoning for human robot interaction". In: *International Conference on Humanoid Robots*. IEEE-RAS. 2009, pp. 331–336.

[76] B. Marthi, S. Russell, and J. Wolfe. "Angelic Semantics for High-Level Actions". In: *ICAPS*. 2007.

[77] D. Mcdermott et al. *PDDL - The Planning Domain Definition Language*. Tech. rep. Yale Center for Computational Vision and Control, 1998.

[78] J. Modayil, T. Bai, and H. Kautz. "Improving the recognition of interleaved activities". In: *Proceedings of the 10th international conference on Ubiquitous computing*. ACM. 2008, pp. 40–43.

[79] T. Mori and S. Scherer. "First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. May 2013, pp. 1750–1757.

[80] S. Mueller and M. Soper. "Microprocessor types and specifications". *InfromIT Network* 3.22 (2006), p. 01.

[81] M. Muneyasu et al. "A realization of edge-preserving smoothing filters using layered neural networks". In: *International Conference on Neural Networks*. Vol. 4. IEEE. 1995, pp. 1903–1906.

[82] R. Munroe. *xkcd: Tasks*. License: https://creativecommons.org/licenses/by-nc/2.5/. 2018. URL: https://xkcd.com/1425/ (visited on 03/27/2018).

[83] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". *Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163.

[84] A. Navada et al. "Overview of use of decision tree algorithms in machine learning". In: *Control and System Graduate Research Colloquium (ICSGRC)*. IEEE. June 2011, pp. 37–42.

[85] M. A. Olivares-Méndez et al. "Fuzzy controller for UAV-landing task using 3D-position visual estimation". In: *International Conference on Fuzzy Systems (FUZZ)*. IEEE. July 2010, pp. 1–8.

[86] M. Pandey, M. Bhatia, and A. Bansal. "An anatomization of noise removal techniques on medical images". In: *International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*. Feb. 2016, pp. 224–229.

[87] S. Papert. *The Summer Vision Project*. Tech. rep. Massachusetts Institute of Technology, Artificial Intelligence Group, July 1966.

[88] J. Papon et al. "Point Cloud Video Object Segmentation using a Persistent Supervoxel World-Model". In: *International Conference on Intelligent Robots and Systems IROS*. IEEE/RSJ. Tokyo (Japan), Nov. 2013, pp. 3712–3718.

[89] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. "Plan guided reaction". *Transactions on Systems, Man and Cybernetics* 20.6 (1990), pp. 1370–1382.

[90] E. P. Pednault. "ADL and the state-transition model of action". *Journal of logic and computation* 4.5 (1994), pp. 467–512.

[91] P. Perona and J. Malik. "Scale-space and edge detection using anisotropic diffusion". *Transactions on pattern analysis and machine intelligence* 12.7 (1990), pp. 629–639.

[92] S. Pinker. *The language instinct: How the mind creates language*. Penguin UK, 2003.

[93] E. Plaku and G. Hager. "Sampling-Based Motion and Symbolic Action Planning with geometric and differential constraints". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. May 2010, pp. 5002–5008.

[94] M. Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software*. 2009.

[95] S. Radiansyah, M. Kusrini, and L. Prasetyo. "Quadcopter applications for wildlife monitoring". In: *IOP Conference Series: Earth and Environmental Science*. Vol. 54. 1. IOP Publishing. 2017, pp. 1–8.

[96] A. Rayes and S. Samer. "Internet of Things—From Hype to Reality". *The road to Digitization. River Publisher Series in Communications, Denmark* 49 (2017).

[97] S. Reich, M. J. Aein, and F. Wörgötter. "Context Dependent Action Affordances and their Execution using an Ontology of Actions and 3D Geometric Reasoning". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp*. Vol. 5. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 218–229.

[98] S. Reich, F. Wörgötter, and B. Dellen. "A Real-Time Edge-Preserving Denoising Filter". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP): Visapp*. Vol. 4. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 85–94.

[99] S. Reich et al. "A Novel Real-time Edge-Preserving Smoothing Filter". In: *Proceedings of the International Conference on Computer Vision Theory and Applications - Volume 1: VISAPP*. Vol. 1. INSTICC. Barcelona (Spain): SciTePress, Feb. 2013, pp. 5–14.

[100] S. Reich et al. "Omnidirectional visual odometry for flying robots using low-power hardware". In: *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applica-*

tions (VISIGRAPP): Visapp. Vol. 5. INSTICC. Funchal, Madeira (Portugal): SciTePress, Jan. 2018, pp. 499–507.

[101] G. Rizzolatti and L. Craighero. "The mirror-neuron system". *Annu. Rev. Neurosci.* 27 (2004), pp. 169–192.

[102] G. R. Rodríguez-Canosa et al. "A real-time method to detect and track moving objects (DATMO) from unmanned aerial vehicles (UAVs) using a single camera". *Remote Sensing* 4.4 (2012), pp. 1090–1111.

[103] M. Rosheim. *Leonardo's Lost Robots*. Springer Science & Business Media, 2006.

[104] B. Rosman and S. Ramamoorthy. "Learning spatial relationships between objects". *The International Journal of Robotics Research* 30.11 (2011), pp. 1328–1342.

[105] E. Rosten and T. Drummond. "Machine Learning for High-Speed Corner Detection". In: *9th European Conference on Computer Vision (ECCV)*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443.

[106] E. Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *International conference on computer vision*. 2011, pp. 2564–2571.

[107] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms". *Physica D: Nonlinear Phenomena* 60.1 (1992), pp. 259–268.

[108] D. Scaramuzza, A. Martinelli, and R. Siegwart. "A flexible technique for accurate omnidirectional camera calibration and structure from motion". In: *International Conference on Computer Vision Systems (ICVS)*. IEEE. 2006, pp. 45–45.

[109] M. Schoeler et al. "Fast self-supervised on-line training for object recognition specifically for robotic applications". In: *International Conference on Computer Vision Theory and Applications*. Vol. 2. IEEE. 2014, pp. 94–103.

[110] A. Schöllig et al. "So You Think You Can Dance? Rhythmic Flight Performances with Quadrocopters". In: *Controls and Art*. Springer International Publishing, 2014, pp. 73–105.

[111] L. Shao et al. "From heuristic optimization to dictionary learning: A review and comprehensive comparison of image denoising algorithms". *Transactions on Cybernetics* 44.7 (2014), pp. 1001–1013.

[112] R. Shaw and J. Bransford. *The theory of affordances; Perceiving, acting and knowing: Toward an ecological psychology*. Vol. 27. Routledge, 2017.

[113] J. Shi and C. Tomasi. "Good features to track". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 1994, pp. 593–600.

[114] K. Sjoo and P. Jensfelt. "Learning spatial relations from functional simulation". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. Sept. 2011, pp. 1513–1519.

[115] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[116] S. C. Stein et al. "Object Partitioning using Local Convexity". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2014, pp. 304–311.

[117] K. Subburaj and B. Ravi. "High resolution medical models and geometric reasoning starting from CT/MR images". In: *International Conference on Computer-Aided Design and Computer Graphics*. IEEE. 2007, pp. 441–444.

[118] K. Subburaj, B. Ravi, and M. Agarwal. "Automated 3D geometric reasoning in computer assisted joint reconstructive surgery". In: *International Conference on Automation Science and Engineering (CASE)*. IEEE. 2009, pp. 367–372.

[119] L. Takayama, W. Ju, and C. Nass. "Beyond dirty, dangerous and dull: what everyday people think robots should do". In: *Proceedings of the 3rd ACM/IEEE International conference on Human robot interaction*. ACM/IEEE. 2008, pp. 25–32.

[120] C. Tomasi and R. Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision*. IEEE. 1998, pp. 839–846.

[121] D. N. N. Tran et al. "Scene recognition in traffic surveillance system using Neural Network and probabilistic model". In: *2017 International Conference on System Science and Engineering (ICSSE)*. July 2017, pp. 226–230.

[122] S. Tregillus, M. Al Zayer, and E. Folmer. "Handsfree Omnidirectional VR Navigation Using Head Tilt". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: ACM, 2017, pp. 4063–4068.

[123] B. Triggs et al. "Bundle adjustment—a modern synthesis". In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[124] Y.-C. Tseng et al. "A wireless human motion capturing system for home rehabilitation". In: *Tenth International Conference on Mobile Data Management: Systems, Services and Middleware (MDM)*. IEEE. 2009, pp. 359–360.

[125] V. Turchin. "The cybernetic ontology of action". *Kybernetes* 22.2 (1993), pp. 10–30.

[126] A. M. Turing. "Computing machinery and intelligence". *Mind* LIX.236 (1950), pp. 433–460.

[127] G. Vásárhelyi et al. "Outdoor flocking and formation flight with autonomous aerial robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 3866–3873.

[128] J. Verbeke et al. "A constraint-based flight control system architecture for UAVs using the iTaSC framework". In: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. June 2016, pp. 310–319.

[129] B. Williams et al. "A comparison of loop closing techniques in monocular SLAM". *Robotics and Autonomous Systems* 57.12 (2009). Inside Data Association, pp. 1188–1197.

[130] M. Winands, Y. Bjornsson, and J.-T. Saito. "Monte Carlo Tree Search in Lines of Action". *Transactions on Computational Intelligence and AI in Games* 2.4 (Dec. 2010), pp. 239–250.

[131] M. Windolf, N. Götzen, and M. Morlock. "Systematic accuracy and precision analysis of video motion capturing systems—exemplified on the Vicon-460 system". *Journal of biomechanics* 41.12 (2008), pp. 2776–2780.

[132] M. Wooldridge and N. R. Jennings. "Intelligent agents: Theory and practice". *The knowledge engineering review* 10.2 (1995), pp. 115–152.

[133] F. Wörgötter et al. "A simple ontology of manipulation actions based on hand-object relations". *Transactions on Autonomous Mental Development* 5.2 (2013), pp. 117–134.

[134] K. Yamane, Y. Yamaguchi, and Y. Nakamura. "Human motion database with a binary tree and node transition graphs". English. *Autonomous Robots* 30.1 (2011), pp. 87–98.

[135] Q. Yang, S. Wang, and N. Ahuja. "Svm for edge-preserving filtering". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2010, pp. 1775–1782.

[136] W. Zhang, M. W. Mueller, and R. D'Andrea. "A controllable flying vehicle with a single moving part". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3275–3281.

[137] Z. Zhang et al. "Benefit of large field-of-view cameras for visual odometry". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 801–808.

[138] Y. Zhu, A. Fathi, and L. Fei-Fei. "Reasoning about object affordances in a knowledge base representation". In: *European conference on computer vision*. Springer. 2014, pp. 408–424.

[139] D. Zoran and Y. Weiss. "From learning models of natural image patches to whole image restoration". In: *International Conference on Computer Vision (ICCV)*. IEEE. 2011, pp. 479–486.

A

Appendices

A.1. Edge-Preserving Filter in the continuous domain

Let $\boldsymbol{f}(\boldsymbol{x})$ define the smoothed input image, $\boldsymbol{h}(\boldsymbol{x})$ the output image, $\boldsymbol{c}(\boldsymbol{\zeta},\ \boldsymbol{x})$ measures the geometric closeness defined by $\boldsymbol{\zeta}$ around $\boldsymbol{x}$), and $\boldsymbol{s}(\boldsymbol{f}(\boldsymbol{\zeta}),\ \boldsymbol{f}(\boldsymbol{x}))$ the photometric similarity. As the filter targets color images, bold letters refer vectors, which also may contain RGB values. In this section $|\cdot|$ also refers to per-element-multiplication instead of vector multiplication. In this approach, noise needs to be detected first based on the user-defined parameter τ. If noise is detected, it should be removed, and in case of a color edge, the edge should be preserved. Therefore, a mean value $\boldsymbol{m}(\boldsymbol{x})$ is defined as:

$$\begin{aligned} \boldsymbol{m}(\boldsymbol{x}) &= \boldsymbol{k}_m^{-1}(\boldsymbol{x}) \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \boldsymbol{f}\left(\boldsymbol{\zeta}\right) \cdot \boldsymbol{c}\left(\boldsymbol{\zeta},\ \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta} \\ \boldsymbol{k}_m(\boldsymbol{x}) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \boldsymbol{c}\left(\boldsymbol{\zeta},\ \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta} \end{aligned} \tag{A.1}$$

and a distance function

$$d\left(\boldsymbol{f}\left(\boldsymbol{x}\right),\ \boldsymbol{m}\left(\boldsymbol{x}\right)\right) = \left|\boldsymbol{f}\left(\boldsymbol{x}\right) - \boldsymbol{m}\left(\boldsymbol{x}\right)\right|, \tag{A.2}$$

which results in the pixelwise distance. The mean value $\boldsymbol{m}(\boldsymbol{x})$ now holds the average color value inside a spatial neighborhood of $\boldsymbol{x}$ and d holds the color distance from the pixel to the average $\boldsymbol{m}(\boldsymbol{x})$. If the spatial neighborhood holds

only small scaled noise, a low pixelwise distance d is expected, as well as a low average pixelwise distance in the spatial neighborhood $\boldsymbol{c}$:

$$\begin{aligned} p\left(\boldsymbol{x}\right) &= k_p^{-1} \iint_{-\infty}^{\infty} d\left(\boldsymbol{f}(\boldsymbol{\zeta}),\, \boldsymbol{m}(\boldsymbol{x})\right) \boldsymbol{c}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta} \\ k_p(\boldsymbol{x}) &= \iint_{-\infty}^{\infty} \boldsymbol{c}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta}. \end{aligned} \tag{A.3}$$

Therefore, the decision can be based on a threshold τ as

$$\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{k}_R^{-1}(\boldsymbol{x}) \cdot \begin{cases} \iint_{-\infty}^{\infty} \boldsymbol{f}\left(\boldsymbol{\zeta}\right) \cdot \boldsymbol{c}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \cdot \boldsymbol{s}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta} & p \leq \tau \\ \iint_{-\infty}^{\infty} \boldsymbol{f}\left(\boldsymbol{\zeta}\right) \cdot \boldsymbol{c}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \cdot \boldsymbol{s}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta} & p \leq \tau, d > \tau \\ \iint_{-\infty}^{\infty} \boldsymbol{f}\left(\boldsymbol{\zeta}\right) \cdot \boldsymbol{c}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) \mathrm{d}\boldsymbol{\zeta} & \text{else,} \end{cases}$$

where k_R is the respective normalization. d can now be used to distinguish large scale noise. A 2d step function

$$\boldsymbol{c}\left(\boldsymbol{\zeta},\, \boldsymbol{x}\right) = \begin{cases} 1 & \boldsymbol{x} - \boldsymbol{a} \leq \boldsymbol{\zeta} \leq \boldsymbol{x} + \boldsymbol{b} \\ 0 & \text{else} \end{cases}, \tag{A.4}$$

holding the conditions $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{e} \in \mathbb{R}_{\geq 0}^2 | \boldsymbol{a} + \boldsymbol{b} = \boldsymbol{e}$ with a fixed $\boldsymbol{e}$ is used. This generates a rectangle of the size $\boldsymbol{e}$ around $\boldsymbol{x}$. As this definition is not feasible in the continuous domain as it generates a non-finite number of subwindows to calculate, in the discrete case however every pixel is checked and updated according to its neighborhood $\boldsymbol{e}$. As a measure for similarity a squared distance is used

$$\boldsymbol{s}\left(\left(\boldsymbol{\zeta}\right),\, \boldsymbol{x}\right) = (\tau - d(\boldsymbol{f}(\boldsymbol{x}),\, \boldsymbol{m}(\boldsymbol{x})))^2 |\boldsymbol{m}(\boldsymbol{x})| \tag{A.5}$$

and the Euclidean norm. In case of noise detection the output is moved to the

mean. The maximum size of the step can be adjusted via the threshold τ.