Aalborg University
Department of Architecture, Design and Media Technology

Master Thesis

Real-Time Image Segmentation Using a Fixation-Based Approach

submitted by Bjarne Großmann

May 30, 2012

AALBORG UNIVERSITET

Aalborg University Copenhagen
Department of Architecture, Design and Media Technology
Lautrupvang 4A
DK-2750 Ballerup
Phone (+45) 9940 9940
http://www.create.aau.dk/

Title
Real-Time Image Segmentation Using a Fixation-Based Approach

Topic
M.Sc. Thesis

Project Period
February 1, 2012 to May 31, 2012

Semester
10th

Author
Bjarne Großmann

Supervisor
Volker Krüger
vok@m-tech.aau.dk

Pages: 120

Finalized: May 30, 2012

Synopsis

The major objective of this thesis is to refine the novel approach of a fixation-based segmentation by Mishra et al. called *Active segmentation with fixation* in oder to enable it to run on a robotic real-time system. In contrast to classical segmentation methods where an image of a scene is divided into multiple individual regions, the fixation-based approach redefines the segmentation process by imitating the human visual system, thus it only separates the fixated region from the rest of the image. The key to this approach is the application of the image segmentation in polar space.

The framework developed in this thesis implements several optimizations and extensions of the original approach. The basic idea is to reduce the strong dependency of the results on the edge detection by introducing an additional optimization step – the *Grab Cut* algorithm. In order to incorporate the *Grab Cut* algorithm, it is examined how to bridge the information gap between the given fixation point and the information needed for the *Grab Cut* to produce correct segmentation results.

The solution is to include the original method as an intermediate step of the new algorithm. This yields a new and more balanced algorithm which can be used in a robotic real-time system.

Abstract

One of the major high-level tasks in computer vision is the process of object detection and recognition. During this process, it is crucial to separate different objects in a visual scene – the classical problem of image segmentation. This thesis presents an algorithm which is based on the novel idea of a fixation-based segmentation by Mishra et al. called *Active segmentation with fixation*. The authors rephrase the general segmentation problem as a binary labeling problem in polar space for a single object. Their result relies highly on the complex computation of a probabilistic boundary map, as it is the crucial point for an effective segmentation. However, due to its computational complexity, the algorithm cannot be used in real-time systems yet.

The major objective of this thesis is to refine this approach in oder to enable the new algorithm to run on a robotic real-time system. Therefore, the framework developed in this thesis implements several optimizations and extensions of the original approach. The basic idea is to reduce the strong dependency of the results on the edge detection by introducing an additional optimization step – the grab cut algorithm.

Based thereupon, the proposed framework has several advantages: By using the output of the graph cut as the input for the grab cut, the framework of Mishra et al. becomes an intermediate step for computing an appropriate input mask. This allows to achieve high-quality results even for erroneous segmentations made by the graph cut. As the grab cut is less dependent on the quality of the edge detection, the computational complexity of this process can be reduced drastically, hence boosting the overall performance of the image segmentation.

The grab cut, being an extension of the graph cut, also profits from all the advantages of the polar space transformation, especially the scale invariance. The polar space itself in this approach is replaced in favor of the log-polar space which not only blurs textures close to the fixation point, but also allows to generate the color models for the grab cut more precisely, as the log-polar representation increases the object region. Moreover, a growing kernel size following the increasing cell-size of the log-polar grid for the edge detection is implemented, thereby imitating an aspect of the human visual system: The blurred vision outside the focus.

Table of Contents

Table of Contents **I**

List of Figures **III**

List of Tables **VII**

1 Introduction **1**

1.1 Interactive Segmentation 6

1.2 Perspective of a Robot 11

1.3 A Fixation-Based Grab Cut Approach in Real-Time 14

1.4 Outline of the Thesis 17

2 Theoretical Foundation **18**

2.1 Edge Detection 19

2.1.1 Gradient Computation by Convolution 19

2.1.2 Kernel Operators 21

2.1.3 Canny Algorithm 25

2.2 (Log-)Polar Space Transformation 26

2.2.1 Polar Space and Log-Polar Space 27

2.2.2 Discretization 29

2.2.3 Rotation and Scale Invariance 30

2.3 Graph Cut and Grab Cut 32

2.3.1 Estimation 34

2.3.2 Markov Random Field 36

2.3.3 Graph Cut 38

2.3.4 Grab Cut 41

3 Segmentation Framework **44**

3.1 Implementational Details 46

3.2 Algorithm and Modules 49

3.3 Fixation Point Selection 51

3.4 Color Space Transformation 52

3.4.1 Standard Color Spaces 53

3.4.2 Hybrid Color Spaces 55

3.4.3 Microsoft Kinect Camera 56

3.4.4 Comparison of the Color Spaces 57

3.5 (Log-)Polar Space Transformation 61

3.5.1 (Log-)Polar Coordinate Mapping 61

3.5.2 Simple Polar Space vs. Log-Polar Space 63

3.5.3 Polar Space and Edge Detection 65

3.5.4 Border Handling 66

3.6 Edge Detection 68

3.6.1 Kernel Design 71

3.6.2 Disparity Edge Map 73

3.6.3 Combining the Edge Maps 74

3.7 Graph Cut 76

3.7.1 Graph Cut in Polar Space 77

3.7.2 Energy Function 78

3.8 Grab Cut 81

3.8.1 Energy Function 83

3.8.2 The ‘Shortcut’ Problem 84

3.9 Contour Detection 86

4 Experimental Evaluation **88**

4.1 Experimental Setup 89

4.1.1 Parameter Setup 89

4.1.2 Scene Setup 92

4.2 General Results 94

4.2.1 Simple Objects 95

4.2.2 Small and Large Objects 96
4.2.3 Objects of Different Shapes 97
4.2.4 Textured Objects 99
4.3 Analysis 101
4.3.1 Disparity Map 101
4.3.2 Log-Polar Transformation 104
4.3.3 Edge Detection 106
4.3.4 Graph Cut and Grab Cut 110
4.3.5 Computational Complexity 113
5 Conclusion and Future Work **117**
Bibliography **IX**

List of Figures

1.1 Fascinating visual system – the human eye is able to recognize different objects fast and easily. 1

1.2 **(a)** A natural image scene. **(b)** and **(c)** The same scene segmented using the Normalized Cut algorithm with different numbers of regions (10 and 60) respectively.[1] 3

1.3 **(a)** A woman sitting on the street. **(b)** The same scene with marked fixation points and saccades. **(c)** The 'puzzle pieces' extracted from the fixation points.[2] 4

1.4 **(a)** The user input for the Magic Wand algorithm. The white regions mark preselected regions. **(b)** The result of the segmentation process of the Magic Wand method.[3] 7

1.5 **(a)** The user input for the Live Wire algorithm. The yellow 'x'es mark the manually added seed points. **(b)** The result of the segmentation process of the Live Wire method.[4] 8

1.6 **(a)** The user input for the graph cut algorithm. The red and white regions mark the preselected regions for background and foreground respectively. **(b)** The result of the segmentation process of the graph cut method.[5] 9

1.7 **(a)** The user input for the grab cut algorithm. The rectangle roughly marks the object of interest. **(b)** The result of the segmentation process of the grab cut method (including border matting of the edges).[6] 10

1.8 **(a)** A complex visual scene of a beach with clouds. **(b)** The corresponding saliency map, as computed by the algorithm in [Itti et al., 1998]. 12

2.1 The Prewitt, the Sobel, the Laplace of Gaussian and the Canny edge detector by comparison. 21

2.2 A point on the polar grid in Cartesian space. It shows the correlation between the Cartesian and polar space. 26

2.3 A comparison of the polar and log-polar grid in Cartesian space. . 27

2.4 The transformation of a region in Cartesian space to the polar grid. 29

2.5 The properties of the polar space transformation. The green and red regions show the scale invariance, whereas the red and the blue region show the rotation invariance of the polar space. 31

2.6 A graph and the corresponding cut. 39

3.1 The graphical user interface at work. The left window shows the video player with its controls at the bottom. The top right window shows the processed image, in this case a resize filter and an edge detection filter was applied. The bottom right window shows the registered parameters which can be changed on the fly. 48

3.2 The different substeps and their arrangement proposed by [Mishra et al., 2009]. 49

3.3 The different substeps and their arrangement proposed in this thesis. 50

3.4 A test image in RGB color space. 54

3.5 A test image in the similar **(a)** Lab and **(b)** YCbCr color space. . 54

3.6 A test image in HSV color space. 55

3.7 A test image in the artificial LHG color space. 56

3.8 **(a)** Contrast-enhanced image retrieved from the Kinect camera in a bright environment. **(b)** Contrast-enhanced image retrieved from the Kinect camera in a dark environment. Many red and blue artifacts appear in the image. 57

3.9 The comparison of the k-means clustering with four components in different color spaces. 58

3.10 The comparison of the k-means clustering (k=5) of chromaticity in different color spaces. 59

3.11 The comparison of the k-means clustering (k=5) of the test image exposed to 10% Gaussian noise. 60

3.12 The effect of the polar space mapping and the remapping to Cartesian space on a mesh. 62

3.13 The difference between the simple polar space and the log-polar space of a transformed star. Already elongated regions get stretched even more in polar space. 63

3.14 The difference between the simple polar space and the log-polar space of transformed squares. The log-polar keeps the aspect ratio of objects. 64

3.15 The difference between the simple polar space and the log-polar space of transformed squares. The log-polar increases the object region and therefore allows to build more precise color models for the grab cut algorithm. 64

3.16 The applied edge detection kernel in log-polar space equals the edge detection with an increasing kernel in Cartesian space. The kernel is oriented along rays emanating from the pole. 65

3.17 Three examples of smooth image edges approximated by a logistic function. 71

3.18 The approximation of the averaged derivation of Gaussian functions by a Gaussian function. 72

3.19 The generation of the implemented edge kernel. Red denotes negative and green positive values. 73

3.20 A disparity map retrieved from the Microsoft Kinect sensor. . . . 74

3.21 **(a)** A disc consisting of two circles with different intensities. **(b)** The corresponding disc in polar space with the pole in the center of the disc. The vertical axis represents the angle, the horizontal axis the radius. 77

3.22 **(a)** A gray star in Cartesian space with the fixation point marked in red. **(b)** The result of the graph cut algorithm in polar space. Note the cut-off spikes of the star. **(c)** The result of the grab cut algorithm with an iteration step. The spikes are now included. . . 84

4.1 The result of different object segmentations composed in a single image. **(a)** shows the segmentation of mostly solid objects with strong depth information. **(b)** shows the segmentation of soft and flat objects with weak depth information. The 'x' marks the chosen fixation point for each object respectively. The objects' contours are marked in green. 94

4.2 A segmentation of simple objects. 95

4.3 A segmentation of simple **(a)** small and **(b)** large objects. 96

4.4 A segmentation of objects with **(a)** elongated and **(b)** more complicated shapes. 98

4.5 A segmentation of textured objects. 99

4.6 The effect of the 'cleaning' procedure of the disparity map. The top row shows the original disparity maps, whereas the bottom row shows the 'cleaned' results respectively. 102

4.7 A segmentation with an incorrect disparity map. The error of the disparity map has no effect on the graph cut algorithm. 103

4.8 A segmentation with an incorrect disparity map. The error of the disparity map is reintroduced by the grab cut algorithm. 103

4.9 The transformation of different sized objects into log-polar space. The similar shapes are both located near the center. 104

4.10 A segmentation of a disc with an internal edge. Due to the scale invariance the disc is segmented correctly. 105

4.11 The effect of different fixation point selections on an elongated object in log-polar space. Depending on the fixation point, parts of the object are cut off. 106

4.12 The effect of different mixing weights for the color and disparity edge map. 107

4.13 The effect of the combined blurring of the edge detector kernel and the log-polar transformation on the graph cut result. The edge maps show from top to bottom: Internal edges due to noise, a small-scaled texture pattern and a large-scaled texture pattern. 109

4.14 The effect of the strong blurring of edges at close quarters which leads to defects in the contour. 110

4.15 The refinement of the graph cut results after applying the grab cut algorithm with five iterations. The top row shows the graph cut results. The middle row shows the grab cut results. The bottom row shows the added regions in green and the subtracted regions in red. 111

4.16 The effects of the selection of different fixation points. **(b)** shows an undesired excluded region due to the fixation point selection. . 112

4.17 The effects of the selection of different fixation points. The object segmentation depends on the context of the fixation points. 113

List of Tables

4.1 The measured absolute and relative duration for the different steps of the proposed algorithm. 114

4.2 The indicated duration for the different steps of the algorithm proposed by [Mishra and Aloimonos, 2011]. 115

4.3 The duration of the implementations as multiples of the graph cut duration and as a factor of the performance gain for the proposed implementation. 116

Chapter 1

Introduction

The human eye is a fascinating visual system – we look around and there we see a horse with its foal standing on a meadow, over there a white stone house with a thatched root and here some trees with green crowns irregularly illuminated by the sunlight (fig. 1.1). For the human visual system, it does not take much effort to distinguish between different objects. Even though the human ability of recognizing different objects in a scene seems like an easy task, it is still an unmatched goal in the area of computer vision.

(a) Horse with its foal **(b)** Thatched house **(c)** Trees in the sun

Figure 1.1: *Fascinating visual system – the human eye is able to recognize different objects fast and easily.*

Recognizing objects in a scene has been a challenge since the beginning of computer vision. However, especially in the last decades where computer systems are used more and more to assist us in our daily life or even to execute various tasks autonomously, the demand for sophisticated detection and recognizing algorithms has been increasing drastically. Nowadays, these algorithms are applied

to many areas of the daily life, e.g. to medical image analysis for detecting tumors or counting cells, face detection on photo cameras, traffic counting, parcel sorting machines, input devices for video games like the *Microsoft Kinect*, 3D reconstruction, motion detection and tracking systems for surveillance or intelligent car guidance systems, just to mention a few [Pham et al., 2000, Coifman, 1998, Viola and Jones, 2004]. However, these systems are far from working as precise as the human visual system such that delicate tasks still have to be supervised and their results have to be verified by a human being. Nonetheless, a lot of ambitious attempts have been made to model a comparable system[1].

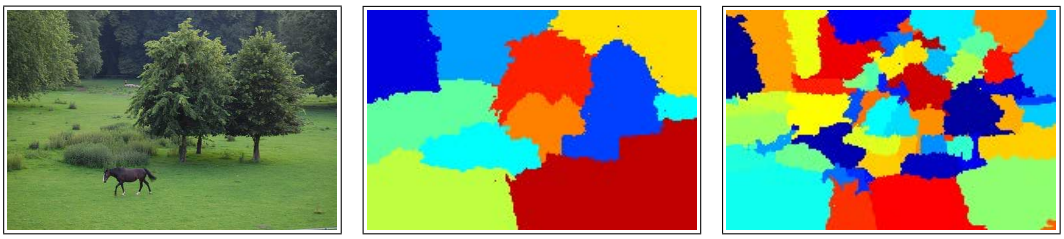
It turned out that a common challenge during the process of object recognition is to find and separate different (object) regions in the scene, since an image only consists of pixels with various values after all. The partitioning of the image into distinct regions made up of connected pixels with similar properties is called image segmentation [Shapiro and Stockman, 2001]. The regions depend much on the use of the application – it can be used to reduce complexity, to separate foreground and background or, like in this case, to find probable object regions.

Regarding this problem, many different approaches[2] have been made and many of them yielded in acceptable solutions, but were often limited to their specific area of interest. Moreover, these approaches often integrate application-related prior knowledge, hence they are far away from presenting a general solution. As a matter of fact, finding a universal approach for image segmentation is not possible – it is an ill-posed problem, since the definition of objects or regions of interest depends on the intention or interest of the viewer or observer [Mishra, 2010].

[1] [Yilmaz et al., 2006,Lim et al., 2009,Malisiewicz and Efros, 2008] present various approaches for object detection and recognition.

[2] See [Cheng et al., 2001] and [Lucchese et al., 2001] for an overview of used techniques and their applications

Fig. 1.2a shows an image which is segmented by a Normalized Cut algorithm with different expected numbers of regions. If the viewer is interested in the trees, fig. 1.2b may be the 'correct' segmentation, or if he is interested in the horse, fig.1.2c may be the appropriate one.



(a) **(b)** **(c)**

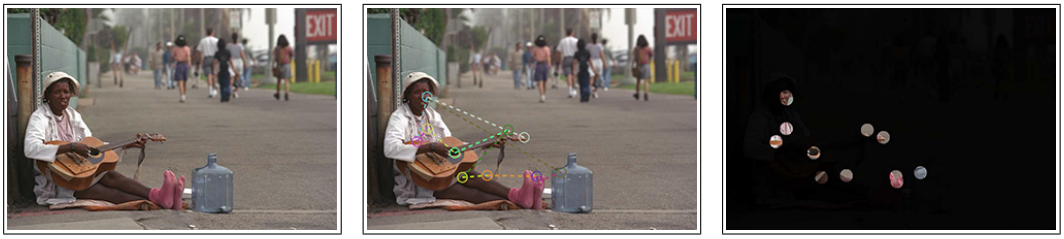
Figure 1.2: ***(a)*** *A natural image scene.* ***(b)*** *and* ***(c)*** *The same scene segmented using the Normalized Cut algorithm with different numbers of regions (10 and 60) respectively.*[3]

But how does the human visual system actually solve this problem? Unlike the traditional image segmentation approaches where all regions are segmented at once, the visual system processes all the visual information step by step[4].

When watching a scene or an image, the human visual system makes a series of automatic short and rapid eye movements called saccades to scan the whole visual scene. Between these saccades, the view focuses on various salient locations called fixation points. Even though the purpose of eye movements, especially regarding the saccades, is not fully understood, it seems that one major reason for this behavior lies in the structure of the human retina. It has the highest concentration of receptors for color information – the so called cones – which are located at the central fovea. There, the eye is able to capture visual information with the highest resolution [Jonas et al., 1992]. Thus, the representation of a scene is built like a puzzle with high-resolution pieces from the fixation points, as visualized in fig. 1.3.

[3]Images taken from [Mishra, 2010]

[4]There has been done a lot of research in investigating and understanding the human visual system. See [Rayner, 1998] for a detailed overview.



(a) **(b)** **(c)**

Figure 1.3: ***(a)*** *A woman sitting on the street.* ***(b)*** *The same scene with marked fixation points and saccades.* ***(c)*** *The 'puzzle pieces' extracted from the fixation points.*[5]

How can this principle of operation be transferred to the segmentation process? The answer to this question is simple: The segmentation should depend on the viewer's intention or interest. The point of interest is to be derived from the functioning of the visual system: As a premise, it is assumed that every fixation point between the saccades can be defined as a location on an object the viewer is interested in. It obviously makes sense, since a viewer would not look at a tree in the scene if he was interested in the horse on the meadow. By only evaluating one step, i.e. one fixation point, it is possible to rephrase the general segmentation problem as a well-posed one, since the region of interest is known to the segmentation algorithm beforehand [Mishra et al., 2009].

So instead of segmenting the whole scene, it is now sufficient to segment the region of interest, which in most cases is the fixated object, by finding its 'optimal' enclosing contour. Yet, knowing the object before segmenting the scene seems to lead the whole process ad absurdum[6], as the segmentation is usually used to recognize objects in the first place. So, how is it possible to identify an object beforehand? Even though it looks like a chicken and egg problem, there exist a lot of approaches to solve this dilemma[7]. Interactive segmentation methods solve

[5]Images taken from [Mishra, 2010]

[6]The are approaches using prior known objects to recognize these objects in the image again, but in general, there is no information given about the objects beforehand.

[7]This can be done by automated systems which are trained on certain objects beforehand [Belongie and Malik, 2000] or by hierarchical multi-scale approaches [Arbelaez et al., 2011]. Another approach are semi-automatic systems based on global parameter settings [Kanungo et al., 2002] or user interaction (section 1.1)

this problem by letting the user choose what object he is interested in, that is by taking more or less user interaction into account.

The framework developed in this thesis picks up on the idea of user interaction for the segmentation process, since finding a solution for the general segmentation problem is impossible. Therefore, section 1.1 presents some of the most popular interactive segmentation methods. Their common goal is to accurately extract a region or object of interest by using the user's or viewer's knowledge about the scene or the objects while minimizing the interaction and response time. Still, the three subgoal accuracy, minimal interaction and response time altogether remain unattained.

However, a much more important question rises from the interactive segmentation approach: Is it possible to apply this concept to a robotic system, since the robot takes the place of the user now? As there is no clear answer to it, this question and possible answers to it are discussed from the perspective of a robotic visual system in section 1.2.

Section 1.3 presents the implemented framework of this thesis which offers a solution to this question. For this purpose, the objective in regard to the use of the proposed segmentation algorithm in a robotic system is defined, as well as the related changes and innovations compared to the original approach of [Mishra et al., 2009].

1.1 Interactive Segmentation

Interactive image segmentation algorithms became more and more popular in recent years, because the problem of automatically segmenting an image or a scene in respect to the viewer's interest is still unsolved – it appears as if the use of human 'hints' is inevitable. Additionally, they became a powerful tool in image editing programs such as *GIMP*[8] or *Adobe Photoshop*[9] where the user input is part of the working process. In the following some state-of-the-art interactive segmentation tools shall be briefly described and compared.

Magic Wand by [Adobe Systems Incorporated, 2012] is one of the best-known selection tools and belongs to the region growing algorithms. The operation method is quite simple: A region of connected pixels is computed by using simple color statistics of the initial point or region with a user adjustable tolerance value. The initial point or region can be expanded, e.g. by using a simple flood-fill algorithm. Due to its simplicity, this approach can be implemented very efficiently, thus resulting in a minimal response time even on older and slower computer systems.

The user interaction is also reduced to a minimum, as one only has to select a pixel or a small region. However, adjusting the tolerance value to a correct level is often cumbersome, if not impossible. The *Magic Wand* can achieve very good results in untextured regions which are bounded by regions of colors outside of the tolerance limit. But in many images the distribution in color of foreground and background pixels can not be distinguished easily and do even overlap sometimes. High textured regions need to have a high tolerance, resulting in unintentionally selected background pixels, or, if the tolerance is set too low, the user has to make his selection almost pixel-wise leaving this tool virtually useless. That is why a

[8] *GIMP* is the *GNU Image Manipulation Program* [GIMP Documentation Team, 2010]. For detail visit: http://www.gimp.org/

[9] The actual version is *Adobe Photoshop CS6* [Adobe Systems Incorporated, 2012]. More informations can be found at http://www.adobe.com/de/products/photoshop.html

clear segmentation of an object with a satisfying contour is often not achieved, as shown in fig. 1.4b.

(a) **(b)**

Figure 1.4: ***(a)*** *The user input for the Magic Wand algorithm. The white regions mark preselected regions.* ***(b)*** *The result of the segmentation process of the Magic Wand method.*[10]

Live Wire (a.k.a. *Intelligent Scissors*), first implemented by Mortensen and Barrett [Mortensen and Barrett, 1995, Mortensen and Barrett, 1998], is an algorithm which is closely related to a manual segmentation using the computerized segmentation as a helping hand. By converting the image to a weighted 2D graph in which the pixels correspond to nodes and the arcs are represented as 8-connectivity neighbors' links, *Live Wire* finds the boundary by implementing the shortest path search in the graph [Shmueli, 2007]. The weights of the graph are based on edges and other boundary-related features, so finding the optimal minimum cost path can be computed by the Dijkstra's algorithm [Dijkstra, 1959] which is proved to be globally optimal. Since the shortest path can only be calculated between two points, this segmentation method is based on strong user interaction.

The user has to select a starting (seed) point and trace the object's boundary roughly. While moving the mouse, the algorithm calculates the minimum cost path between the current mouse position and the (last) seed point and snaps

[10]Images taken from [Rother et al., 2004]

on the nearest strong edges. During this process new seed points are added automatically (called boundary cooling), but if the calculated path deviates from the desired one, the user can set additional seed points to correct and fix the path.

A common application using *Live Wire* is *Adobe Photoshop* where this tool is called *Magnetic Lasso* [Adobe Systems Incorporated, 2012] which is used in the example seen in fig. 1.5a. Even though the result is quite accurate, this approach has one main drawback: The already strong user interaction will be additionally increased in highly textured regions with many strong edges or in 'flat' regions where there are no edges, since in this case various 'minimal' paths are possible.

(a) **(b)**

Figure 1.5: ***(a)*** *The user input for the Live Wire algorithm. The yellow 'x'es mark the manually added seed points.* ***(b)*** *The result of the segmentation process of the Live Wire method.*[11]

Graph Cut is a powerful optimization technique which, as the name already indicates, is based on weighted graphs and can be applied on images as well [Greig et al., 1989, Boykov and Jolly, 2001]. In the graph cut algorithm the image to be inspected is represented as a 2D weighted graph like in the Live Wire technique. Every node in this graph can be in one of two states: Either it is an object

[11] Images taken from [Rother et al., 2004]

pixel or a background pixel. Therefore an internal probabilistic color model[12] is needed, which is initialized based on the user input. The weights on the links are defined by an energy function which consists of both region and boundary information. The approach uses the min-cut/max-flow algorithm to find a binary (object/background) segmentation of the image.

The user has to mark some areas as object and others as background, which in most interfaces can be easily done using wide brush strokes. Often there is no need to define many regions as long as the marked regions allow to build separable color models for the object or background respectively. In case of overlapping color distributions as the image in fig. 1.6 shows, it is sometimes necessary to mark regions as object regions manually, especially at thin boundary regions.

(a) **(b)**

Figure 1.6: ***(a)*** *The user input for the graph cut algorithm. The red and white regions mark the preselected regions for background and foreground respectively.* ***(b)*** *The result of the segmentation process of the graph cut method.*[13]

Grab Cut is an enhancement on the original graph gut algorithm proposed by [Rother et al., 2004]. Instead of using simple color models like histograms, the grab cut algorithm makes use of Gaussian Mixture Models (GMMs) [Reynolds, 2008] to represent foreground and background colors. But the major enhance-

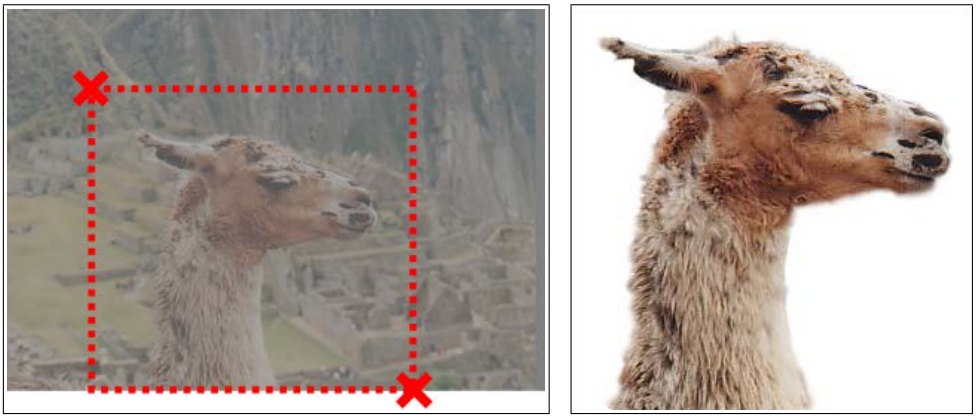
[12] Originally, the graph cut algorithm was used only with binary images [Greig et al., 1989], but was extended to work with gray-scale and color images, too

[13] Images taken from [Rother et al., 2004]

ments on the graph cut are firstly the 'iterative estimation' of the object region and secondly the possibility to make an 'incomplete labeling'. The iterative estimation is done by relearning the GMMs and relabeling the nodes on each iteration and thus refining the solution. Furthermore, the user has the possibility to refine the result manually after each iteration.

The second enhancement, the incomplete labeling, allows the user to define pixels or regions of an image as foreground, background or unlabeled which reduces the degree of user interaction for a given quality of result (see fig. 1.7). Supplementary, it allows the user to initialize the grab cut by simply drawing a rectangle around the desired object, thereby only indicating the background area and reducing the user interaction.

A major drawback of all graph and grab cut algorithms is the fact that they tend to produce small contours by trying to find a minimum cut and using shortcuts especially on thin objects. Nowadays, also interactive methods using progressive graph and grab cuts as image editing tools exist like *Adobe Photoshop* Quick Select or Paint Select [Adobe Systems Incorporated, 2012].



(a) **(b)**

Figure 1.7: ***(a)*** *The user input for the grab cut algorithm. The rectangle roughly marks the object of interest.* ***(b)*** *The result of the segmentation process of the grab cut method (including border matting of the edges).*[14]

[14]Images taken from [Rother et al., 2004]

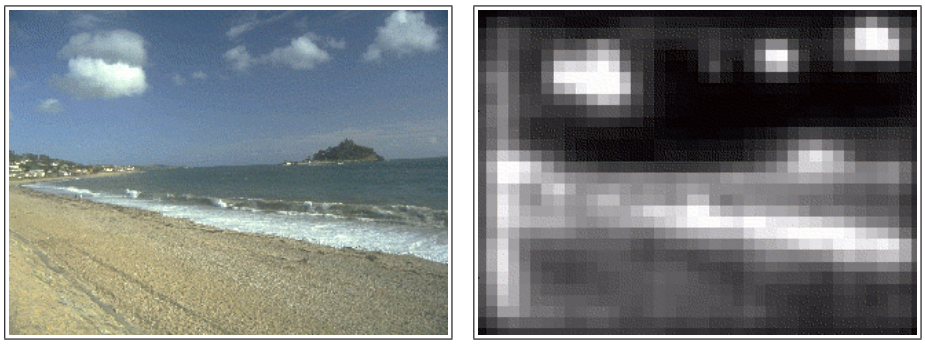
1.2 Perspective of a Robot

Artificial visual systems are an integral part in state-of-the-art robotics. Robot vision and computer vision have a lot in common, however, there are some demands in which they differ after all. In robotics, vision is a means to an end; it is used to allow the robot to interact with its environment and complete various tasks. Therefore, the success of a vision system in a robot is related to its actions and the success of fulfilling the task and cannot be considered as an independent component, but as a part of the whole robotic system [Kyrki and Kragic, 2011]. Unlike in traditional computer vision, it is not sufficient to identify an object in a scene, as the robot has to identify its form and location, e.g. in order to grasp an object. In recent years a lot of advances in areas such as stereo and 3D reconstruction have been made, which form a solid and well-defined theoretical basis for tasks in a 3D environment. A current example of a practical advancement made trough theoretical work is the *Microsoft Kinect* camera system, an inexpensive 3D vision sensor.

As the use of a robot is, in the first instance, practically motivated, there is another important aspect to be taken into consideration in robot vision: the response time. The visual system is only a part of the processing chain and therefore it should work as fast as possible. A robot watching a scene for half an hour before being able to interact makes no sense for practical reasons. So reducing the processing time to a minimum is another subgoal for implementing a robot vision system. Taking the human visual system as a reference, the response time of a robot's visual system should be at least quite similar to that. Depending on the object (especially in case of an unknown object, where there is no prior knowledge about it), in real-time it can take up to some seconds to segment or identify it.

Even though the goals of robot vision and computer vision may differ, the first step of image processing is the same: the image segmentation. So the question rises, why not to implement the same segmentation algorithms mentioned before

in a robot system? The answer is quite obvious: A robot is not a human. The interactive segmentation methods are working fine and give high quality results with human beings, since the user knows the object of interest and has the control over the refinement of the intermediate solutions. If the user is a robot though, it leads to the same chicken and egg problem which led to the introduction of the interactive methods in the first place. That is why the question arises anew: How is it possible to identify an object before segmenting it without relying on any user input? The answer remains the same: It is not possible without a hint of the viewer's interest.



(a) **(b)**

Figure 1.8: ***(a)*** *A complex visual scene of a beach with clouds.* ***(b)*** *The corresponding saliency map, as computed by the algorithm in [Itti et al., 1998].*

But when reducing the information about the object of interest to a minimum, more precisely to a single point of interest on the object, instead of relying on more complex information like contours or background and foreground areas, the interaction is also reduced to the simple selection of a point. In fact, by reducing the user interaction to a minimum, the problem becomes now solvable. Obtaining the point of interest is actually possible by using visual attention systems [Itti et al., 1998]. They can predict locations in the scene that attract the most attention by employing low-level visual cues or by using statistical information about the human visual system. These algorithms often result in so called 'saliency maps' as shown in fig. 1.8, where more attracting regions have higher values than less attracting. Another more manual method is to actively help the robot find an object in the scene by showing it to the robot. To do so, markers or gesture recognition methods can be used.

In the end, for the segmentation process, it does not matter by which fixation strategy the point of interest is retrieved, the only problem left is to infer from a given point on an object to its contour or region. So on the one hand, there is an advanced algorithm like grab cut, which needs not much, but some user knowledge about the object region. In case of the grab cut algorithm, this information does not even need to be overly exact. On the other hand, there is a given point on the object which is insufficient information for a high-level algorithm. So the question is how to bridge this information gap in a robot system autonomously, that is without the interaction of a user?

1.3 A Fixation-Based Grab Cut Approach in Real-Time

This thesis presents a segmentation algorithm which is based on the novel idea of a fixation-based segmentation by [Mishra et al., 2009] called *Active segmentation with fixation*. It answers the question of how to obtain sufficient information about a given point on an object for the graph cut algorithm autonomously such that the algorithm can be applied in a robotic system.

Therefore, the algorithm of [Mishra et al., 2009] is briefly described: The idea behind this approach is to rephrase the general segmentation problem as a binary labeling problem in polar space for a single object. Hence, the user interaction can be reduced to a minimum, more precisely to the selection of a single point, the fixation point. Now, the input of the framework is this point which is located on the object of interest.

In the first step, the calculation of an edge map or probabilistic boundary map is made by using different visual cues. The resulting probabilistic boundary map contains the probability of an edge pixel being on the desired object's boundary. The edge map is then transformed into the polar space where it is used as the input for the second step: the graph cut algorithm. This algorithm uses the edge map by mapping the intensity values to the weights of the links between the nodes of the graph. Then, a path through the edge map is created by computing the minimum cut of the graph. The path is the 'optimal' division of object and background pixels, i.e. the path itself defines the 'optimal' contour of the object in polar space in regard to the edge map.

The graph cut algorithm is carried out a second time, now the weights being adjusted by using color histograms of the labeled areas. The result of the graph cut algorithm is a binary mask with areas labeled as 'inside' or 'outside' the object. After the transformation back to Cartesian space, the 'optimal' contour of the object can easily be extracted from this mask.

The main disadvantage of this approach is that the result relies highly on the complex computation of a probabilistic boundary map, as this is the crucial point for a good segmentation. Due to its computational complexity, the algorithm cannot be used in real-time systems, yet.

The major objective of this thesis is to refine this approach in order to enable the algorithm to run on a robotic real-time system. Therefore, the framework developed in this thesis implements several optimizations and extensions on the original approach. The contributions to the algorithm developed by [Mishra et al., 2009] can be summarized as follows:

- This framework provides an additional grab cut module for the segmentation result given by the approach of [Mishra et al., 2009] which thereby becomes an intermediate step of the whole algorithm. This leads to a more balanced process, as the dependency of the final result does not solely rely on the edge detection. By distributing the liability more evenly on the algorithm, a foundation for supplementary optimization processes can be established, as minor errors occurring in the intermediate steps can be absorbed by the following grab cut algorithm.

- The operation time of the algorithm is reduced for the use in a real-time robot system. As the developed algorithm is very robust to erroneous intermediate results, especially the edge detection step can be optimized a lot by simplifying the low-level routines. This allows the reduction of the computational complexity of the whole algorithm, while the high quality of the algorithm's results nearly remains the same.

- The polar space representation for the contour retrieval is exchanged in favor of the log-polar space. Besides the useful scale invariance property of the polar space, this transformation has two additional advantages: On the one hand, the image gets smoothed near the object of interest which results in less frequent intensity changes of the texture in the object region. This enhances the results of the edge detection, as it will find less edges

near the object caused by texturing. On the other hand, the object region in log-polar space is much larger than in polar space. This enhances the results of the grab cut algorithm, as more pixels are included to build the color models which are used to refine the object's contour.

- The edge detector implements a kernel with a variable size: The kernel size grows with the distance to the object, thereby imitates the human visual system – the blurred vision outside the focus. This implies that the edge detection is more sensible to possible edges near the object than to edges further away where the kernel covers a larger area for computing the edge.

1.4 Outline of the Thesis

The present thesis describes a newly developed approach to image segmentation based on the previous work of [Mishra et al., 2009]. The developed framework implementing this approach relies on a combination of three different image processing techniques: The edge detection, the polar space transformation and the graph/grab cut algorithm. The theoretical background for these methods is explained in chapter 2.

In addition to extending the approach of [Mishra et al., 2009] with the grab cut method, this framework serves as an experimentation and evaluation application for the proposed algorithm, which is described in section 3.1. A major part of this thesis deals with the different substeps of the algorithm, their configuration, implementation and the effects on the final result. Section 3.2 covers the structure and the design of the framework delimits and illustrates the similarities and differences between the framework and the approach of [Mishra et al., 2009]. A detailed view on the different modules involved in the algorithm, including their requirements, implementation and expected effects of parameter settings on the result, is presented in sections 3.3 to 3.8.

Chapter 4 explains the experimental setup, including the parameter settings and the scene setup in section 4.1 and their results on the basis of particular object properties in section 4.2. In section 4.3, the detailed analysis of the proposed algorithm and its modules as well as the computational complexity are covered .

Finally, chapter 5 consists of the conclusion and future work of this thesis, evaluating the developed algorithm and stimulating discussion on its further development. Moreover, the results of the algorithm and the analysis are used as starting points for the further development of this approach.

Chapter 2

Theoretical Foundation

The proposed framework is based on different techniques which are often used and discussed in computer vision. When combining them into one algorithm for image segmentation, it is necessary to understand the basics of the related topics. The segmentation process itself and closely related works were already introduced in section 1.1 to define the objective of this thesis. This chapter presents and discusses the theory that forms the foundation for the major components used for implementing the proposed segmentation algorithm.

One major component in this framework is the edge detection. It plays an important role for the subsequent processing of the image by the graph cut algorithm and holds the greatest potential for optimization in regard to the implementation of [Mishra et al., 2009] (see also chapter 3). Section 2.1 describes the basics of edge detection by convolution and, based upon that, the still state-of-the-art Canny Edge Detection algorithm. Section 2.2 explains the fundamentals of the fixation-based approach by introducing the (log-)polar transformation, the associated (log-)polar space and its properties. The last section 2.3 deals with the graph-based representation of an image which can be used to solve the problem of energy minimization (in regard to finding optimal object contours) by using the graph cut algorithm. Finally, a variation of the graph cut algorithm, namely the grab cut algorithm, is presented.

2.1 Edge Detection

Extracting boundaries from (color) images has been a research area for a long time in computer and machine vision and is a fundamental tool of low-level image processing. The latter often is a necessary step in high-level image processing like image segmentation or detecting objects in a scene [Nadernejad and Sharifzadeh, 2008]. The basic idea behind edge detection is to find relevant changes of color intensity, whereas strong changes indicate regions of different objects. This is the ideal conception of image edges, but in reality, there are a lot of intensity variations caused by changes of the environment like illumination and shading or by characteristics of surfaces like texture and reflectance. Additionally, there can be object boundaries which lack of strong change in intensity like objects of the same color or blurred edges. Moreover, when working with images of lower quality like video camera images, it is inevitable to deal with noise in these images. In the last decades, many approaches have been proposed in order to find an appropriate and reliable edge detector. Even though there might not exist the optimal edge detector for general purpose, many algorithms generate a reasonable output which can be used for further processing.

2.1.1 Gradient Computation by Convolution

Taking the image as a continuous intensity function, finding a sudden change of intensity is achieved by computing the gradient of the function. Therefore, the first derivative in each direction is computed, as strong changes have a high gradient value. The gradient of the image intensity is given by

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) \tag{2.1}$$

The result is a vector which visually points to the steepest increase of intensity at any location. The square magnitude of the gradient is given by

$$\|\nabla I\|^2 = \left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2 \tag{2.2}$$

When using this approach for images, a discretization step has to be done, since the pixels are not continuous. By doing so, it is possible to express the derivative as a convolution kernel, a $(m \times n)$-matrix G, which is then used to calculate a new value for each pixel by calculating the weighted sum of all neighbors given by the matrix. So, by using the finite difference approximation, the first derivatives become

$$\begin{aligned} \frac{\partial I}{\partial x} &\approx I(x+1, y) - I(x-1, y) \\ \frac{\partial I}{\partial y} &\approx I(x, y+1) - I(x, y-1) \end{aligned} \tag{2.3}$$

They can be used to easily calculate the gradient magnitude. By transferring the coefficients of the derivatives as elements of the kernel matrix, the convolution process itself can be expressed as:

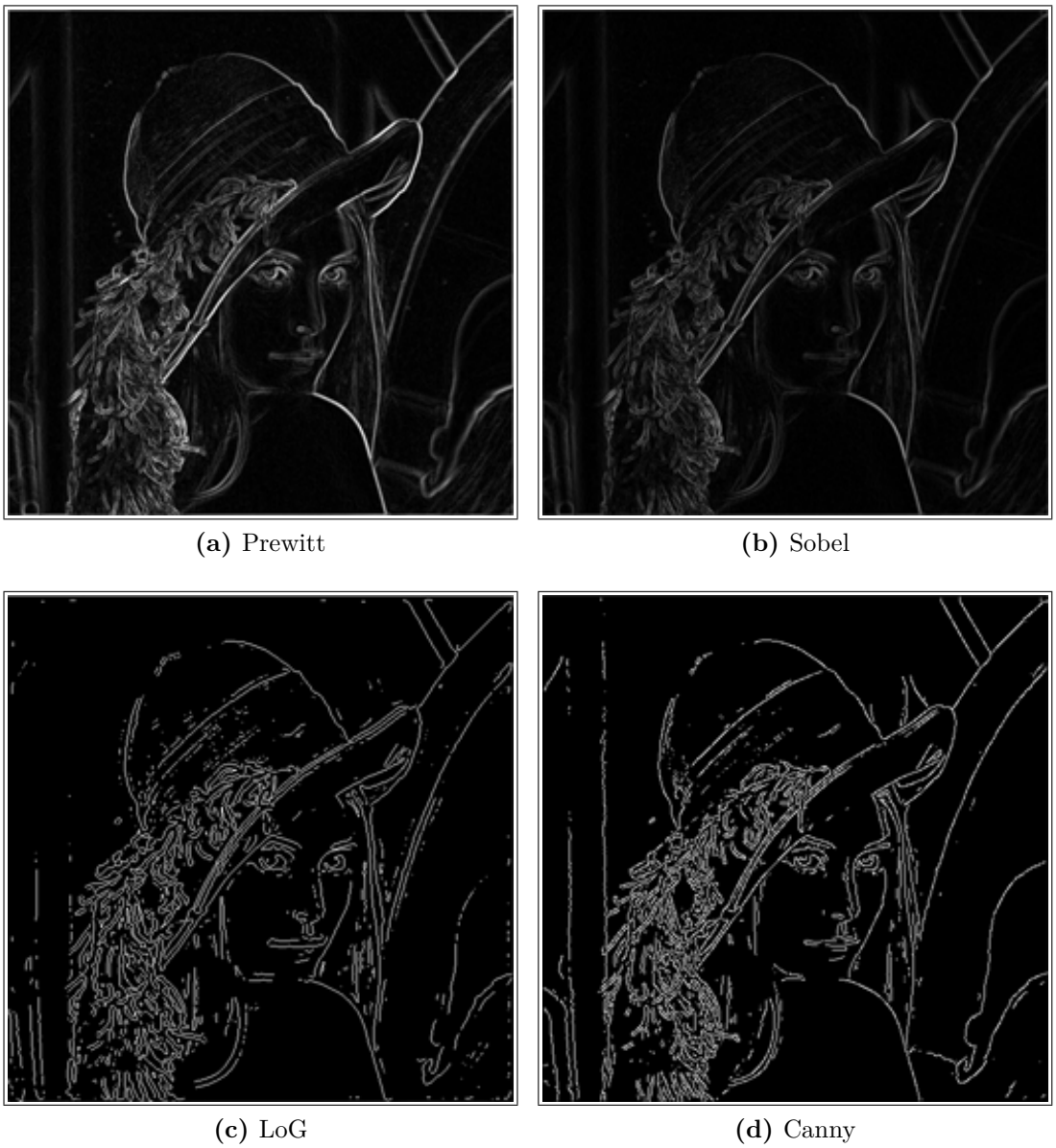
$$\begin{aligned} I_{new}(x, y) &= w(x, y) \otimes I_{old}(x, y) \\ &= \sum_{s=-\frac{m}{2}}^{\frac{m}{2}} \sum_{t=-\frac{n}{2}}^{\frac{n}{2}} w(s, t) \cdot I_{old}(x-s, y-t) \end{aligned} \tag{2.4}$$

where $w(s, t)$ denotes the weights in the matrix.

In most cases, the kernel size is only (3×3) or (5×5). But when using much larger kernels ($\gtrapprox (12 \times 12)$), the convolution needs a lot of computational time. However, it is possible to counteract this by transferring the problem into the frequency domain. For this purpose, the image and the kernel have to be transformed by the Fourier Transformation. In the frequency domain, the convolution step is then reduced to a simple multiplication [Arfken and Weber, 1985].

2.1.2 Kernel Operators

Most classical methods use 2-dimensional convolution kernels for detecting edges in an image. The design of the convolution kernel can vary depending on what kind of edge characteristics it is supposed to respond to, e.g. the direction of the edge or the robustness of the kernel against noise. The most common kernels, as shown in fig. fig:foundation:edge, are presented in this section.



(a) Prewitt **(b)** Sobel

(c) LoG **(d)** Canny

Figure 2.1: *The Prewitt, the Sobel, the Laplace of Gaussian and the Canny edge detector by comparison.*

The Prewitt operator is exactly the kernel described above. The coefficients of the derivatives can be written as:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.5}$$

The filter itself is separable[1] and is used to convolve the image in the horizontal and vertical direction. Therefore, the computation is relatively inexpensive compared to filter kernels without this property. On the other hand, the calculation of the gradient is relatively coarse, especially in high frequency regions of the image or when it is exposed to noise. Therefore, sometimes a (4×4)-filter kernel is used with $G_x = \begin{bmatrix} 3 & 1 & -1 & -3 \end{bmatrix}$ and $G_y = {G_x}^T$. The kernel is slightly more robust to high frequency changes, but has no real center pixel, since it is an even-sized kernel. Fig. 2.1a shows an image of Lena[2] convolved with a Prewitt filter.

The Sobel filter tries to improve the Prewitt filter by removing some of the noise in images using a smoothing filter. This is done by using a triangular or Gaussian kernel before convolving with the edge detection kernel. The Gaussian (3×3)-kernel is defined as

$$G = G_x \cdot G_y = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.6}$$

[1] It can be split into two operations by representing the matrix as a vector multiplication.

[2] Lena is a wide-spread image in the branch of image processing.

By combining the vertical Gaussian and the horizontal Prewitt kernel and the other way around, it is possible to create a new kernel which merges the smoothing and gradient operation into one

$$G_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.7}$$

and

$$G_y = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.8}$$

The Sobel filter is still an often used approach due to its computational performance and reasonable results. Fig. 2.1b shows an image of Lena convolved with a Sobel filter. But even though the Gaussian Blur is applied, the result still suffers from high frequency noises. Additionally, the filter is proved to be not rotationally symmetric, i.e. the filter is more sensitive to horizontal and vertical edges than to edges differing from these angles. Therefore, Hanno Scharr derived a more rotation-invariant (3×3)-filter called Scharr operator or filter [Scharr, 2000].

$$G_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} \quad G_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \tag{2.9}$$

The Laplacian of Gaussians (LoG) filter had been quite famous before Canny developed the 'optimal' filter (see section 2.1.3). This approach was first introduced by [Marr and Hildreth, 1980], who combined the discrete Laplace operator with the Gaussian function. Therefore, it is also known as the Marr-Hildreth operator. The filter also became famous under its nickname 'Mexican Hat' or 'Sombrero', as its visual shape looks like a Mexican sombrero turned upside down. This approach uses the Laplacian operator as the kernel is rotationally

invariant. The Laplace operator is defined as the divergence of the gradient of a function.

$$\Delta f = div\,(gradf) = \nabla \cdot (\nabla f) = \nabla^2 f \tag{2.10}$$

In discrete image space the approximated kernel

$$G = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{or} \quad G = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.11}$$

is used. The latter includes the diagonal edges as well. By using the second derivative of the image intensity function, edges can now be identified by finding the zero-crossings of the applied Laplacian. This involves the additional use of an algorithm to recognize true zero-crossings, since regions with constant intensity result in zero-values as well.

The smoothing of the image before convolving it with a kernel is done by calculating the Laplacian of the Gaussian function. The resulting function is still rotationally invariant, but is much more resistant against high frequency changes. Fig. 2.1c shows an image of Lena convolved with a LoG filter.

The computation of the second derivative of the Gaussian function can be computational complex. However there is an easy way to approximate the LoG filter by using the difference of two Gaussian functions with different σ, also known as the **Difference of Gaussians (DoG)**. The result is almost the same as the one achieved with the LoG filter. An interesting property of the LoG or DoG after detecting the zero crossings is that all found edges are closed contours. This property is often used for blob[3] detecting algorithms.

[3]For the definition of blobs see [Lindeberg, 1993].

2.1.3 Canny Algorithm

The Canny Edge Detector is still one of the state-of-the-art edge detectors, even though John F. Canny has already developed it in 1986 [Canny, 1986]. By using a computational approach, he describes an 'optimal' edge regarding three important criteria for an edge detector:

- Good detection: Edges in the image should not be missed and spurious responses should be minimized.
- Good localization: The distance between marked edges and true edges should be minimized.
- Minimal response: The number of responses to an edge in the image should be minimized. In the optimal case, an edge is only marked once.

Using these criteria, Canny proposes a multi-stage algorithm of which the first step is the noise reduction by smoothing the image with a Gaussian kernel. The second step is the detection of edges using a oriented edge detection operator like Prewitt or Sobel for each direction. Both filters can be approximated by applying the derivative kernel on the Gaussian. This combination is the first derivative of the Gaussian function. In the third step, the edge direction angles are used to apply a non-maximum suppression which finds the local maximum of the gradient magnitude along the gradient direction. The output of this process is an edge image with thinned edges. In the last step, the detection of true edges and the removal of irrelevant edges is done by hysteresis thresholding: A lower and an upper threshold determine if an edge is a true edge. The ones above the upper threshold are taken as genuine edges and the ones below the lower threshold are dropped. The edges in between will be taken if they are connected to a genuine edge. This can be realized by using the genuine edges as a starting point and then tracing the edge throughout the image with the directional information provided by the gradient computation. Fig. 2.1d shows an image of Lena convolved with a Canny edge detector.

2.2 (Log-)Polar Space Transformation

The use of the (log-)polar space in pattern recognition and computer vision has increased more and more over the recent years. The reason for this is that it has proved advantages over uniformly sampled spaces like the Cartesian space. The greatest benefits gained from (log-)polar transformed images are their rotational and scale invariance properties. Many mathematical formulations needed for various vision tasks can be simplified because the rotational- and scale-dependent transformations can be reduced to a simple vector addition [Araujo and Dias, 1997]. Another advantage of the (log-)polar transformation is data reduction by decreasing the resolution at the image periphery, as shown in fig. 2.2.

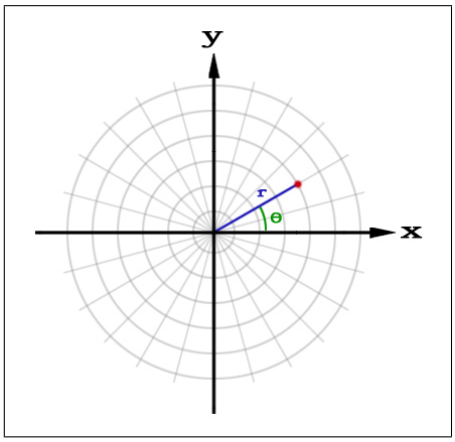


Figure 2.2: *A point on the polar grid in Cartesian space. It shows the correlation between the Cartesian and polar space.*

These advantages are exploited in many approaches including in active vision systems, in binocular tracking systems or in motion recovery and estimation [Schwartz and Greve, 1995, Bernardino and Santos-Victor, 1999, Tistarelli and Sandini, 1993]. Using the polar space for image segmentation is a very new idea. From what is known, the fixation-based approach developed by [Mishra et al., 2009] in 2009 is the first framework to implement a segmentation based on graph cuts using the polar space. Since this thesis is based on this idea, it is important to understand the (log-)polar mapping and its properties. Therefore, this section presents the layout of the (log-)polar space and the mathematical formulation behind.

2.2.1 Polar Space and Log-Polar Space

The (log-)polar coordinate system is a two dimensional coordinate system. The origin of this system is called 'pole' and is an equivalent to the origin of the Cartesian system. Each point (r, θ) in the coordinate plane can be described by a distance from the pole called 'radial coordinate' or radius r. θ denotes the angle, which is called 'angular coordinate' or polar angle, from a fixed direction. It is possible to convert any given point (r, θ) from polar to Cartesian coordinates (x, y) by

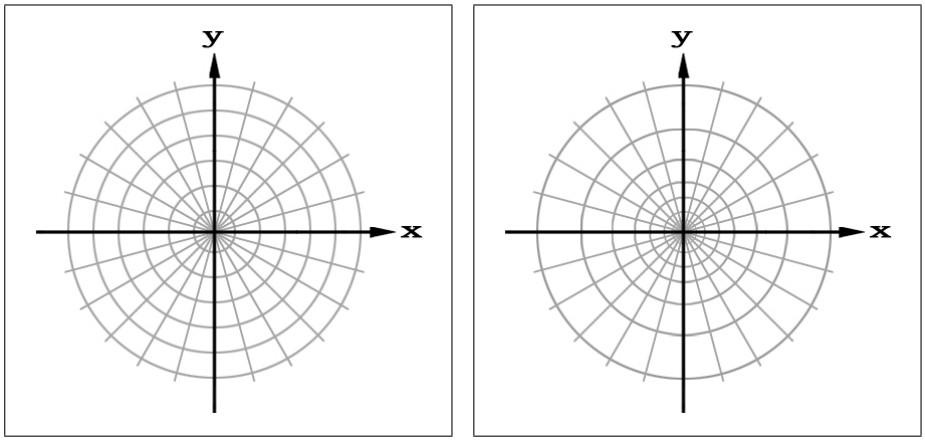
$$x = r\cos\theta \quad \text{and} \quad y = r\sin\theta \tag{2.12}$$

or vice versa by the inverse transformation

$$r = \sqrt{(y^2 + x^2)} \quad \text{and} \quad \theta = \text{atan2}\,(x, y) \tag{2.13}$$

where atan2 denotes the variation of the arctan function by taking the respective coordinate quadrants into account.

Fig. 2.3a shows polar grid with the correlation between the Cartesian and the polar space. An interesting property of this grid is the size of the grid cells: The radius is increasing constantly from the pole, whereas the distance of two points lying on two adjacent rays at a certain radius grows exponentially with the radius.



(a) Polar space **(b)** Log-polar space

Figure 2.3: *A comparison of the polar and log-polar grid in Cartesian space.*

It has the effect, that grid cells further away from the pole seem to be compressed along the radial axis. This distortion is equivalent to the projection of a visual scene on the human retina as determined by [Schwartz, 1984]. This effect occurs due to concave-shaped retina plane within the eye. The human brain tries to compensate this distortion by transforming the retinal image into its cortical projection. According to Schwartz, this mapping can be approximated by a logarithmic function. Therefore, for the transformation into Cartesian space, the inverse function, more precisely the exponential function, has to be applied on the radius, which results in

$$x = \exp(r) \cdot \cos\theta \quad \text{and} \quad y = \exp(r) \cdot \sin\theta \tag{2.14}$$

and the logarithmic function[4] for computing the radius in polar space

$$r = \log\left(\sqrt{(y^2 + x^2)}\right) \quad \text{and} \quad \theta = \operatorname{atan2}(x, y) \tag{2.15}$$

This formulation often helps to simplify mathematic problems in analytics e.g. the Laplace's equation. In computer vision, the use of the log-polar transformation is more canonical, even though its properties regarding the rotational and scale invariance are the same. A comparison between the log-polar and the simple polar space is shown in fig. 2.3. For better understanding, the name 'polar space' in this section refers to both, the log- and the polar space, unless not stated otherwise.

[4]The term log denotes the natural logarithmic function throughout this thesis, as it is usually used that way in programming languages.

2.2.2 Discretization

In computer vision, the transformation from and to polar space has to be discretized for practical reasons. In most cases, the source space and the destination space are given in the form of an image. Therefore, the polar space is defined by aligning the polar coordinates into an orthogonal space with the axes r and θ as seen in the right image of fig. 2.4. In the present thesis, the angle θ is represented by the vertical axis, whereas the horizontal axis denotes the radius r. This implies that the radius and the angle become discretized to integers within a range of 0 and a predefined maximum value.

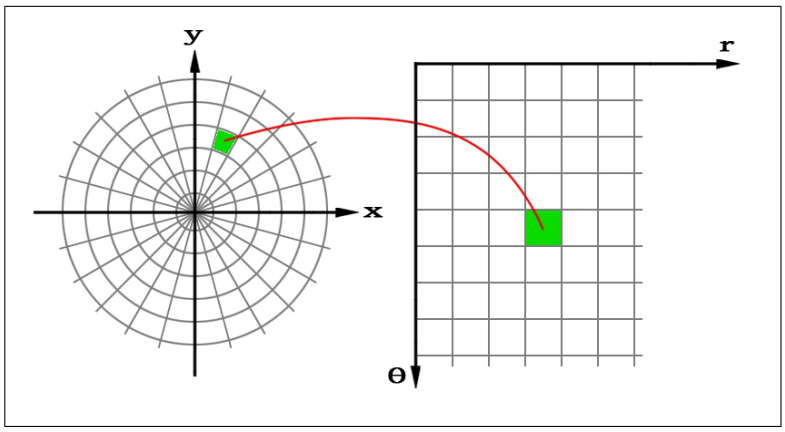


Figure 2.4: *The transformation of a region in Cartesian space to the polar grid.*

When transforming from Cartesian coordinates $(x, y) \in \mathbb{N}^0$ given in the source image, the result are polar coordinates $(r, \theta) \in \mathbb{R}$. Since the destination image is a discrete grid, the values of the polar coordinates have to be rounded to a certain cell of the grid. For example, by taking 360 grid rows, more precisely 360 θ values, the resulting angle has to be rounded to an integral angular degree. The problem arising from this forward transformation is that some of the coordinates in the destination image might not be defined as cells can be skipped due the resolution of the destination grid or the rounding process. Therefore, the usual way of mapping coordinates in discrete space is done by the inverse transformation starting with the destination image. By applying the Cartesian transformation for each cell in the polar space, the results are floating Cartesian coordinates. These coordinates can be interpolated by the neighboring grid cells using the

nearest neighbor, the linear or cubic interpolation method. This procedure has to be carried out for the Cartesian-to-polar transformation, too.

The interpolation of pixel values in the transformed image means a loss of information. The polar space transformed image needs to interpolate many pixel values near the pole, whereas regions more distant to the pole get compressed. When applying the Cartesian transformation again, this results in image artifacts increasing with the distance to the pole, as the compressed region has to be stretched again by interpolation. Decreasing the artifacts can be done by sub-sampling mechanisms or by increasing the resolution of the polar grid which is often unwanted, since the image in the polar space is used for further image processing

2.2.3 Rotation and Scale Invariance

The main reason for transforming an image into polar space is are the scale and rotation invariance properties of the transformed image. This proved to be especially useful for pattern or object recognition tasks, since it drastically minimizes the search domain. The mathematical derivation of these properties can be found in [Alan and II, 2007]. But by looking at different regions transformed from Cartesian space to polar space, it quickly becomes clear that these are valid properties of the polar space, as seen in fig. 2.5.

A circle with a radius r_1 around a point in the Cartesian space, it can be transformed into polar space. As for each polar angle θ, the radius is the same, the function $f(\theta) = r_1$ is a constant line. Another circle around the same point with the radius r_2 is a constant line as well. The only difference is a shift along the radius axis. Hence, scaling in Cartesian space results in a shift along the r-axis in polar space, i.e. it is scale invariant.

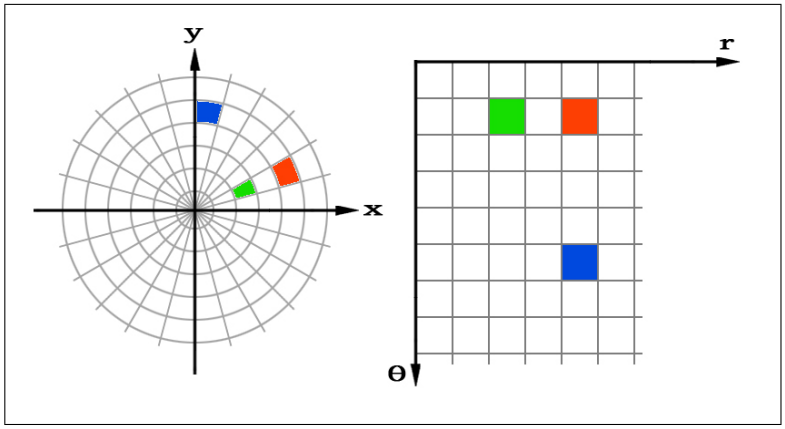


Figure 2.5: *The properties of the polar space transformation. The green and red regions show the scale invariance, whereas the red and the blue region show the rotation invariance of the polar space.*

The rotation invariance can be seen analogously, only taking place on the θ axis. A point in Cartesian space lying on a ray with an angle θ_1 and a copy of that point rotated around the pole lying on a ray with an angle θ_2, then both points have the same distance to the pole, more precisely, the same radius r_3. The function for θ_1 and θ_2 results in $f(\theta_1) = f(\theta_2) = r_3$. Therefore, the only variable that changes is the angle and the rotation becomes a shift along the θ-axis.

2.3 Graph Cut and Grab Cut

The theory of graphs has been a mathematical branch ever since Leonhard Euler tried to find a tour over the seven bridges of the city of Königsberg without crossing a bridge twice [Adams, 2011]. Down to the present day, many problems in daily life or in other mathematical disciplines have been transferred to graph-related problems. One of these problems is to find the minimum cut in a graph, i.e. to find a set of edges with the smallest sum of weights which are dividing the graph into two parts. Calculating the cut had been computationally very expensive until in 1956, when [Ford and Fulkerson, 1956] showed that this calculation can be transferred to the problem of finding the maximum flow in a graph network, today also known as the max-flow/min-cut-theorem.

The maximum flow problem asks for the maximum flow in a graph from a given source to a sink, whereby the edges have different capacities. Figuratively, this problem can be seen as pushing more and more water through a network of pipes until the capacity of the pipes is reached or finding the maximum possible amount of cars driving from a location A to a location B on the traffic network. The theorem states that a maximum flow from a source to a sink through a flow graph saturates a set of edges which are dividing the graph into two partitions. These set of edges correspond to the edges gained by the minimum cut. In fact, the cost of the cut is also equal to the maximum flow capacity.

Ford and Fulkerson also proposed an algorithm to compute the solution [Ford and Fulkerson, 1957] which is called Ford–Fulkerson algorithm. By finding the maximum flow or minimum cut, the graph is partitioned into two distinct parts with different labels.

Not long after this accomplishment, researchers applied this approach to combinatorial optimization problems, as the minimum cut of a graph can be used in order to find an optimal solution for submodular function minimization [Edmonds, 1970]. This concept of energy minimization was taken on by [Greig et al., 1989] in 1989, who first implemented a max-flow algorithm for binary image

restauration. Since the computational complexity was still too high for the computer systems[5], it took some time until it was universalized. In 2001, [Boykov and Jolly, 2001] finally introduced the interactive graph cut approach for image segmentation, also implementing the max-flow algorithm which nowadays can be computed in real-time due to the rapid development of computer hardware.

The proof of optimality for applying the function minimization for binary image labeling was shown by [Greig et al., 1989], as the minimum cut of a graph splits it only into two parts with different labels. Extensions of the graph cut algorithm are nevertheless used for multi-labeling problems which give good approximation of the solution. However, in many cases in computer vision, it is sufficient to use the graph cut algorithm for binary labeling, as e.g. in the proposed framework which uses the different labels for denoting background and foreground.

In order to understand how the binary labeling problem can be solved by a graph-based approach, in the first instance, it is necessary to get the basic idea of the labeling process from the viewpoint of estimation which is discussed in section 2.3.1. Afterwards, in section 2.3.2, an important graph-based representation of probability dependencies, the Markov Random Field (MRF), is described. It is the generalized form of the Ising model which uses the Gibbs probability measure and the related energy function from the field of ferromagnetism to represent a model for estimation. The newly formulated probability function is then used in the graph cut algorithm as explained in section 2.3.3. The section also explains the construction of a flow graph based on an image and briefly presents the Ford-Fulkerson algorithm to find the minimum cut. Finally, section 2.3.4 shows how the graph cut algorithm can be extended to obtain the grab cut algorithm.

[5]They applied the algorithm on an image of size 88×100. The basic algorithm took about 3000 seconds to compute the solution on an Amdahl 470 computer. They proposed a variant, which could be computed in 'only' 250 seconds.

2.3.1 Estimation

Binary labeling is a well-known subproblem of segmentation in general. One classical example is the restauration of a binary image which has been exposed to noise. The task is to find the most probable original image. So, the idea behind binary labeling is to assign a binary label to each element in a given set based on some given constraints or properties. Thereby, the procedure estimates a configuration over all elements which results in an approximate or even exact[6] solution of the posed problem depending on the algorithm used. This problem is in general called an estimation problem which can be solved by an estimator.

An estimator is a function $\hat{\theta}$ to describe the process of estimation. The estimation relies on a given quantity of observed data $X = (x_1, \ldots, x_n)$ and the unobserved population parameter $\theta = (\theta_1, \ldots, \theta_n)$. An estimator detects the maximum a posteriori[7] (MAP) probability $P(\theta|X)$ which is a central term of Bayes' theorem [Bishop, 2007]. It can be written as

$$\hat{\theta}(X) = \arg\max_{\theta} P(\theta|X) \tag{2.16}$$

where the Bayes' theorem formulates

$$P(\theta|X) = \frac{P(X|\theta) \cdot P(\theta)}{P(X)} \tag{2.17}$$

$P(X|\theta)$ is the conditional probability or often called the likelihood of a sample belonging to a certain class and $P(\theta)$ denotes the prior probability based on the prior knowledge. $P(X)$ is the evidence which is the sum of all posterior probabilities θ given the sample X. It can be seen as a normalization constant.

[6] The meaning of 'exact' in this context has to be seen in respect to the formulation of the constraints.

[7] It is also called a maximum likelihood estimator (MLE) if the prior probability is neglected.

The records $(x_1, \ldots, x_n)$ of sample X are conditionally independent and each has its own likelihood function $f(x_i|\theta_i)$. Thus, the joint probability function $l(X, \theta)$ can be written as

$$l(X, \theta) = \prod_{i=0}^{n} f(x_i|\theta_i) \tag{2.18}$$

Often, the likelihood is also written as the log-likelihood[8], as the log function is a monotone increasing function and thereby does not change the result of the maximization. A major advantage of the log-likelihood is that the product can now be written as a simple sum:

$$\log l(X, \theta) = \sum_{i=0}^{n} \log f(x_i|\theta_i) \tag{2.19}$$

The conditional probability for each element of the sample X can vary from application to application. In computer vision, estimators are used in many different contexts and the computation of the probabilities depends much on the parameter models used. This can be average intensity values, color histograms or Gaussian Mixture Models, just to mention a few. Therefore, different similarity measures like the simple difference, Euclidean and Mahalanobis distance, Chi-square distance or the maximum likelihood are applied as well. Often, estimators are used to estimate certain parameters, e.g. the mean and variance of a Gaussian probability function which are used to build color models. This ansatz is persued in simple applications such as binary thresholding or in more sophisticated techniques like the expectation maximization algorithm. In image segmentation, e.g., whole label configurations are estimated to restore a noisy image or split an image into distinct regions [Cheng et al., 2001].

[8]The term log denotes the natural logarithmic function throughout this thesis, as it is usually used that way in programming languages.

The posterior probability regarding the joint probabilities can now be written as

$$\begin{aligned} P(\theta|X) &= \frac{P(X|\theta) \cdot P(\theta)}{P(X)} \\ &\propto P(X|\theta) + P(\theta) \\ &\propto \log P(X|\theta) + \log P(\theta) \\ &\propto \sum_{i=0}^{n} \log P(x_i|\theta_i) + \log P(\theta) \end{aligned} \tag{2.20}$$

This formulation is useful when modeling the prior probability with a Markov Random Field which is used in the formulation of the graph cut algorithm, as it is shown in section 2.3.

2.3.2 Markov Random Field

An advantageous approach to model and compute prior probabilities is the Markov Random Field. It is a convenient way to describe estimation problems based on different configuration states, where the state of a pixel or a site only depends on the state of its neighboring sites. The dependencies between neighboring sites is a prior knowledge and has to be included in the formulation of the problem. In most image segmentation tasks, the interdependency of pixels can be a valid assumption, as single pixels would not differ too much from the surrounding ones. [!REFHRASE argumentation the other way around]

The Markov Random Field describes these dependencies in an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of nodes and $\mathcal{E} = \{e_1, \ldots, e_m\}$ a set of edges connecting the vertices. Given a set of random variables $X = \{X_v\}_{v \in \mathcal{V}}$, a certain assignment for the random variables is called a configuration $x = (x_1, \ldots, x_n) \in X$. The Graph $\mathcal{G}$ is a Markov Random Field if the probability of the random variables X_v can be calculated by the neighborhood $\mathcal{N}_v \subset \mathcal{G}$,

typically four or eight directly connected neighbors. Any graph is called a Markov Random Field, if it satisfies the Markov property

$$X_a \perp\!\!\!\perp X_{\mathcal{V}\setminus\mathcal{N}_a\cup\{v\}} \mid X_{\mathcal{N}_a} \tag{2.21}$$

It states that a variable is conditionally independent of all other variables only given its neighbors.

Due to the Hammerslay-Clifford theorem [Hammersley and Clifford, 1971], the probability[9] of a configuration x can be given by the Gibbs measure:

$$P\left(X = x\right) = \frac{1}{Z\left(\beta\right)} \exp\left(-\beta U\left(x\right)\right) \tag{2.22}$$

where $Z\left(\beta\right)$ is the normalizing partition function and $U\left(x\right)$ the energy of the configuration x. The parameter β is a free parameter[10].

The energy function of the Markov Random Field is thereby defined as a pairwise interaction of the site x_i and its neighbors x_k, where $\mathcal{C} = \{\mathcal{N}_1, \ldots, \mathcal{N}_c\}$ denotes the set of all neighbor sets of configuration x:

$$\begin{aligned} P\left(X = x\right) &\propto \exp\left(-\beta \sum_{\mathcal{N}_c \in \mathcal{C}} V_{\mathcal{N}_c}\left(x\right)\right) \\ &\propto \exp\left(-\beta \sum_{(i,k)} V_{ik}\left(x_i, x_k\right)\right) \end{aligned} \tag{2.23}$$

$V_{\mathcal{N}_c}\left(x\right)$ denoting the clique potential. The exact formulation depends on the application and the underlying models used. Often, the potential function is used to compute some kind of similarity or distance between x_i and $x_k \in \mathcal{N}_c$.

An extension to the Markov Random Field is the Conditional Random Field. It alters the clique potential by incorporating the observed configuration y into the term and thereby conditioning it to the observation. This actually contradicts

[9] Note, that in the general estimation section the random variables are denoted by θ instead of X. This is actually a common convention in literature.

[10] In physics, it is the inverse temperature.

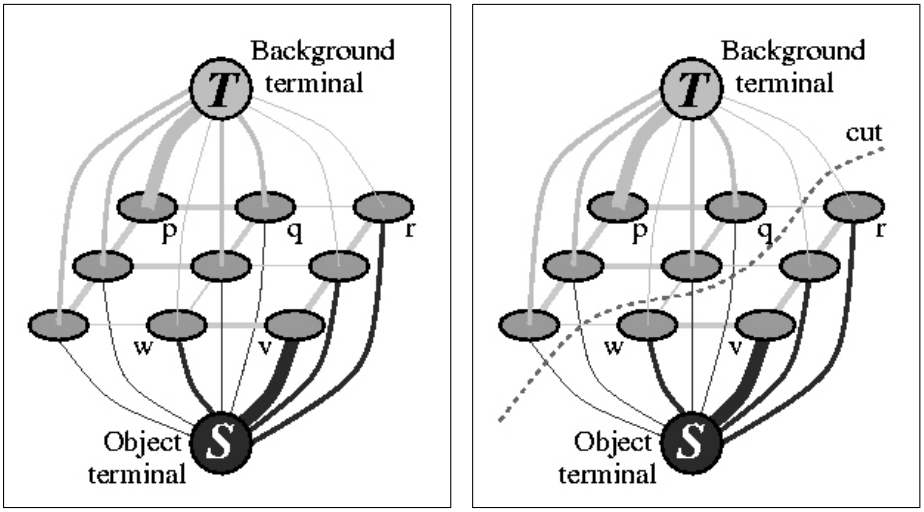
the idea of using the Markov Random Field as a model for the prior, since the prior should not depend on the observation. Therefore, this probability has to be rewritten into the conditional probability, since the computation of $P(Y|X)$ is the likelihood. A major drawback of the Markov Random Field is that the penalty for the distribution of a certain site increases with the difference to its neighbors regardless of the observed data. That means, even if the 'true' data contains a large difference between two neighbors, e.g. the pixels of an edge in an image, the clique potential will still penalize it. The Conditional Random Field fills the information gap by comparing the current configuration with the observed data and by weighting the penalty accordingly. Formally, the Conditional Random Field can be written as

$$P(Y|X) \propto \exp\left(-\beta \sum_{(i,k)} V_{ik}(x_i, x_k, y_i, y_k)\right) \tag{2.24}$$

2.3.3 Graph Cut

The graph cut algorithm is based on the prior probability formulation of the Markov Random Field or Conditional Random Field. It is a maximum a posterior estimator which uses an underlying graph for modeling and evaluating the posterior probability. As the graph cut algorithm is used for binary labeling, the solution is 'exact', i.e. it finds the configuration which globally maximizes the posterior probability.

The graph cut algorithm is based on a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ which consists of nodes $\mathcal{V} = \mathcal{I} \cup \{s, t\}$ directly connected by bidirectional weighted edges $\mathcal{E}$. The nodes correlate to the pixels of an image $\mathcal{I}$, except two additional terminal nodes $\{s, t\}$ which are called source s and sink t. Each graph edge is assigned some non-negative weight, cost or energy $w(p, q)$, where the inverse direction $w(q, p)$ can have a different weight. Edges connecting only nodes from $\mathcal{I}$ are called n-links, whereas the edges from nodes to terminal nodes are called t-links. Nodes of $\mathcal{I}$ are only connected to their direct neighbors $\mathcal{N}$ and to both terminal nodes, as seen in fig. 2.6a.



(a) A graph. **(b)** A cut through the graph.

Figure 2.6: *A graph and the corresponding cut.*

As stated in section 2.3, the minimum cut or min-cut of this graph can then be used to minimize certain energy functions. A cut partitions the graph into two distinct subsets $\mathcal{S}$ and $\mathcal{T}$ such that the terminal nodes s and t are elements of the sets $\mathcal{S}$ and $\mathcal{T}$ respectively. The cut is therefore also called s-t-cut. Fig. 2.6b shows such a cut through a graph.

A cutset $\mathcal{C}_i \in \mathcal{C}$ is a set of edges (p, q) such that $p \in \mathcal{S}$ and $q \in \mathcal{T}$. The cost $|C_i|$ of a cut is the sum of the weights or costs w_{pq} over the edges in the cutset $\mathcal{C}_i$. The minimum cut problem is to find the cut with the minimum costs among all cuts.

$$|C_{min}| = \arg\min_{\mathcal{C}} |C_i| \tag{2.25}$$

with

$$|C_i| = \sum_{(p,q)\in\mathcal{C}_i} w_{pq} \tag{2.26}$$

Furthermore, the edges in $\mathcal{C}_i$ can be split into t-links and n-links. When making a cut, for each node p the cut contains a t-link, either the edge (p, s) or (p, t). Additionally, the cut can also contain n-links connecting p with its neighbor q, be it that $p \in \mathcal{S}$ and $q \in \mathcal{T}$. The selection of the t-link (p, s) or (p, t) at node p

can be modeled using a predicate $D_p(f_p)$ for selecting (p, s) or (p, t) respectively. Therefore, the cost of the cut can now be written as

$$|C_i| = \sum_{p \in \mathcal{I}} D_p(f_p) + \sum_{\substack{(p,q) \in \mathcal{N} \\ p \in \mathcal{S}, q \in \mathcal{T}}} w_{pq} \tag{2.27}$$

This formulation is a special case of an energy function. It can be expressed in a the generalized form

$$E(X) = \sum_{p \in \mathcal{I}} U_p(x_p) + \lambda \sum_{(p,q) \in \mathcal{N}} V_{p,q} \cdot \delta(x_p, x_q) \tag{2.28}$$

with λ denoting a weighting constant between the regional term or data constraint as well as the boundary term or regularizing constraint and the additional submodularity constraint [Kolmogorov and Zabih, 2004]

$$V_{pq}(0,0) + V_{pq}(1,1) \leq V_{pq}(0,1) + V_{pq}(1,0) \tag{2.29}$$

included as

$$\delta(x_p, x_q) = \begin{cases} 1 & \text{if } x_p \neq x_q \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

The idea of the graph cut algorithm is to represent the maximum a posteriori probability with such an energy function. It can then be minimized by computing the minimum cut or more precisely the maximum flow with the Ford-Fulkerson algorithm. The energy function can be derived by formulating the MAP estimator with the help of the Markov Random Field or Conditional Random Field. The posterior probability can be written as

$$\begin{aligned} P(\theta|X) &\propto \sum_{p \in \mathcal{I}} \log P(x_p|\theta_p) + \sum_{i=0}^{n} \log P(\theta_p) \\ &\propto \sum_{p \in \mathcal{I}} \log P(x_p|\theta_p) + \left(-\beta \sum_{(p,q) \in \mathcal{N}} V_{pq}(\theta_p, \theta_q) \right) \end{aligned} \tag{2.31}$$

The posterior probability can now be used in the MAP estimator. But instead of maximizing the probability, the problem is rephrased as a minimization problem by multiplying the term with -1.

$$\begin{aligned}\hat{\theta}_{min}(X,\theta) &= \arg\min_{\theta} -P(\theta|X) \\ &\propto \arg\min_{\theta} \sum_{p\in\mathcal{I}} -\log P(x_p|\theta_p) - \left(-\beta \sum_{(p,q)\in\mathcal{N}} V_{pq}(\theta_p,\theta_q)\right) \\ &\propto \arg\min_{\theta} \sum_{p\in\mathcal{I}} U_p(\theta_p) + \lambda \sum_{(p,q)\in\mathcal{N}} V_{pq}(\theta_p,\theta_q) \end{aligned} \tag{2.32}$$

When the submodularity $\delta(\theta_p,\theta_q)$ for the boundary term is integrated, the last formulation is equivalent to the energy function which can be minimized via the min-cut-max-flow algorithm.

2.3.4 Grab Cut

The grab cut algorithm is based on the graph cut definition. The major difference is the formulation of the energy function's regional term. The graph cut approach in its original version, as described by [Rother et al., 2004], implements an estimator for binary labeling, whereas the likelihood probability only depends on the observed pixel values.

Recalling the binary labeling problem on binary images (e.g. image restauration) [Greig et al., 1989], it can informally be described as follows: A binary image exposed to noise is given as a configuration of binary labels. The aim is to find the 'true' image, the estimate. Each pixel label is estimated by the observed image and its direct neighbors. The regional term returns a constant weight in case the observed value and the tested configuration differ, elsewise it returns a even smaller constant weight or 0. The boundary term returns the sum of the potentials between the current pixel and its neighbors. The potential is given by a constant weight when the neighbor value differs from the current pixel value, otherwise 0. The minimum cut can be used to find an exact minimum configuration by using the max-flow algorithm.

When it comes to image segmentation, the observed pixel values are often intensity or color values. These values have to be mapped to a binary map, such that the resulting mask can be used as the input for the graph cut algorithm. Therefore, prior knowledge of the image has to be incorporated in order to create an initial labeling. This is often done by user interaction as described in section 1.1, whereat the labeling is not complete. Hence, the initial labeling is used to estimate two intensity or color representations[11] of the image, for each label respectively. By comparing the observed pixel values to the models, the regional term can now be evaluated. In terms of the minimum cut algorithm, this process is equivalent to the computation of the weights for the t-links.

Using color models has a major impact on the graph cut algorithm as a MAP estimator: As the initial labels are based on incomplete user input or other selection methods, the whole estimation process additionally relies on the parameters of the color models. E.g., when using a gray-scale image for the binary labeling problem for binary images, the gray-scale image has to be converted into a binary representation. Given a color model for the original image as a mean value, the binary image can be derived by simple thresholding. When using more sophisticated color models, the similarity measure between the models and the observed color value has to be adapted, e.g. by using the Euclidean distance or the negative log-likelihood. Instead of mapping the color image to a binary representation, this process can implicitly be achieved by estimating the weights of the t-links. Therefore, depending on the observed pixel value, the weights of the t-links (p, s) and (p, t) denote the energy of 'keeping' or 'switching' its label respectively to the color models. This means that the formulation of a color model also introduces another estimator which has to be implemented. Since the color models are learned from the image used in the labeling process, the result depends on the parameters of the models which are above all latent variables

[11] In the following, intensity and color representations will be generally denoted as color models.

of the estimation. Estimation errors of the color model estimator can then get multiplied in the graph cut algorithm.

Regarding these difficulties, the grab cut algorithm enhances the graph cut algorithm. Firstly, it proposes a more sophisticated color model, namely the Gaussian Mixture Model (GMM), which is less error-prone when classifying colors. It describes multiple sub-populations as Gaussian probability functions within an overall population [Bishop, 2007]. The initial learning of the different Gaussians in this model has to be carried out with a set of training samples. In fact, this process is another estimation and is often done using an expectation-maximization (EM) algorithm. In case of the implementation of [Rother et al., 2004], the initialization of the Gaussians is accomplished by using the k-means algorithm. EM algorithms are often applied to estimation problems, where the estimate depends on unobserved latent variables.

Having said that, the second improvement of the grab cut algorithm is quite obvious: As the grab cut estimation also depends on the latent color model parameters, the whole estimation process can be seen as an EM algorithm. Therefore, [Rother et al., 2004] proposes to refine the color models by an iterative use of the graph cut algorithm. The estimation by the graph cut can be seen as an expectation step, whereas the labeling of the pixel is the maximization step. The labeled pixels are then used to improve the GMM color models. After certain iterations, the color models converge. The grab cut algorithm supplementary allows users to refine the result by marking additional foreground or background regions between each iteration step in order to reduce errors and to achieve a faster convergence.

Chapter 3

Segmentation Framework

The implementation of this framework is an extension of the approach from Mishra, Aloimonos and Fah [Mishra et al., 2009] as described in section 1.3. The algorithm input is an image of a scene (additionally a disparity map when using the *Microsoft Kinect* camera) and an user selected point on an object of interest. The image(s) are then transformed to (log-)polar space. The edge detection computes an edge (probability) map from the given image(s). Optionally, this can also be done before the (log-)polar transformation (see section 3.5.3). Afterwards, the grab cut algorithm finds an approximate solution for the object's contour as a labeled map which is used as the input for the grab cut algorithm. It iteratively refines the segmentation by using the edge map and a sophisticated color model. The result is the contour of the fixated object in the scene.

The algorithm proposed by [Mishra et al., 2009] gives good results on simple objects, but cannot be run in real-time. As it is an important objective of this framework, it is necessary to reduce the computational complexity by optimizing the different step of their algorithm. The idea is to find a suitable balance between quality and performance while reducing the user interaction to a minimum, more precisely to a simple selection of a fixation point (cf. section 1.2).

Before optimizing the algorithm, it is necessary to identify the computational bottleneck in regard to the optimization process. In [Mishra and Aloimonos, 2011],

the researchers extended their approach by introducing an automated fixation strategy where they also mention the computational complexity. Their algorithm is not able to be used as a real-time system[1], as the edge detection itself already needs 6 seconds to compute. Computing the optical flow map (taking 24 seconds) can be disregarded because in the proposed framework the disparity map of a *Microsoft Kinect* camera is used instead (which takes about 1sec). The segmentation of a given fixation point takes about 2 seconds. Further computations made in [Mishra and Aloimonos, 2011] can also be neglected due to the method of using multiple fixation points. Clearly, the edge detection algorithm is the bottleneck of the segmentation framework where most of the optimization can be realized.

Increasing the performance of the edge detection implies the quality reduction of the edge map. That in turn raises the error ratio of the graph cut algorithm, especially for regions with blurry contours. Therefore, the outcome of simplifying the edge detection module is as expected: It will work much better on simple objects with clearly defined borders than on objects occurring e.g. in natural scenes. But the proposed framework is designed to work in robot systems identifying objects to interact with, therefore the simplification is reasonable.

Even though the quality may suffer from the optimization during the edge detection process, it can partly be compensated by the new approach of this framework. The grab cut algorithm, used in the end to segment the image, can handle certain degree of errors in the edge map by using multiple iterations and sophisticated color models. This approach is described in section 3.8.

The following sections discuss the general design of the segmentation framework, the reasons for the specific implementation, the discussion on the parameter selection and its anticipated results. The first section 3.1 describes the framework itself, the technical implementation as well as the interface design, whereas the second section 3.2 gives a overview of the different modules involved. Afterwards

[1]Their time analysis has been done using a quad-core 32-bit processor.

in sections 3.3 to 3.9, each module is presented on its own with its requirements, specification, implementation and effects on the final result.

3.1 Implementational Details

The implementation of the proposed segmentation framework considers three major aspects for the evaluation of the experimental results:

- The framework is based on a state-of-the-art computer vision library for efficient and fast algorithms, especially regarding the usage in a robotic real-time system.
- The interface provides an optimal visual control enabling a fast evaluation. It allows the change of parameters for the different substeps of the algorithm on the fly, so that there is no need for compiling and restarting the application for each little change of the parameters.
- The implemented components are built modularly for chaining them with little effort. This provides a maximum of flexibility and reusability for evaluating different module arrangements or experimenting with different implementations of visual cues. Image processing components can be easily added and included in the chain. Additionally, the modules are independent from the graphical user interface such that they can be used as a library in other applications, i.e. as a node in a robotic system based on *ROS*[2].

For implementing the framework, the well-known computer vision library *OpenCV*[3] is used. It is a state-of-the-art computer vision library providing a wide range of efficiently implemented image processing and manipulation algorithms. These

[2] *ROS* is the abbreviation for *Robot Operating System* which is commonly used in robot vision.

[3] See http://opencv.willowgarage.com for more information. Detailed descriptions and explanations can be found in [Bradski and Kaehler, 2008].

make use of the internal computer hardware structure, thereby additionally boosting the execution time. This is an important factor for optimizing the algorithms in regard to the use in a robotic real-time system. Moreover, it allows to focus on the rapid development of the different modules by using these available tools. An interesting recent feature of *OpenCV* is the interface to the *Robot Operating System ROS*. It provides a bridge between these two libraries and allows an easy transformation to the *ROS* image representation and vice versa.

The implementation of the modules makes excessive use of the *OpenCV* library methods, especially in low-level algorithms. The modules consist of one or multiple image filters. A filter is defined as an image-based operation or transformation function which gets a list of input images and generates one or multiple output images. The programming interface provides an easy way to concatenate or 'chain' these in- and outputs, thereby allowing to build complex and exchangeable filter chains. Each filter can explicitly register variables which can be changed via the graphical user interface for a simple change of parameter values on the fly for a faster evaluation.

Besides the filter system, the framework provides an easy access method to images, videos and camera devices as a video resource. The output of this wrapping class can be directly used by the filter chain. Furthermore, it allows to define different modes of playback for maximum control: Each video resource can be played frame by frame by proceeding to the next image manually. Another mode proceeds to the next image automatically when the filtering process is done. The third method is a real-time playback of the video resource. This mode actually simulates a camera device for videos, i.e. the video and the image capture process runs in an own thread, while the image processing procedure takes place in another thread. The image processing thread gets only the actual captured image, thereby skips multiple frames depending on the duration of the image segmentation. The parallel implementation of image retrieval and processing is done mainly for performance reasons: The image grabbing routine takes some time as it has to access the video file or the camera device. Therefore, it would

be better to already grab the image before the segmentation instead of grabbing it when it is actually needed.

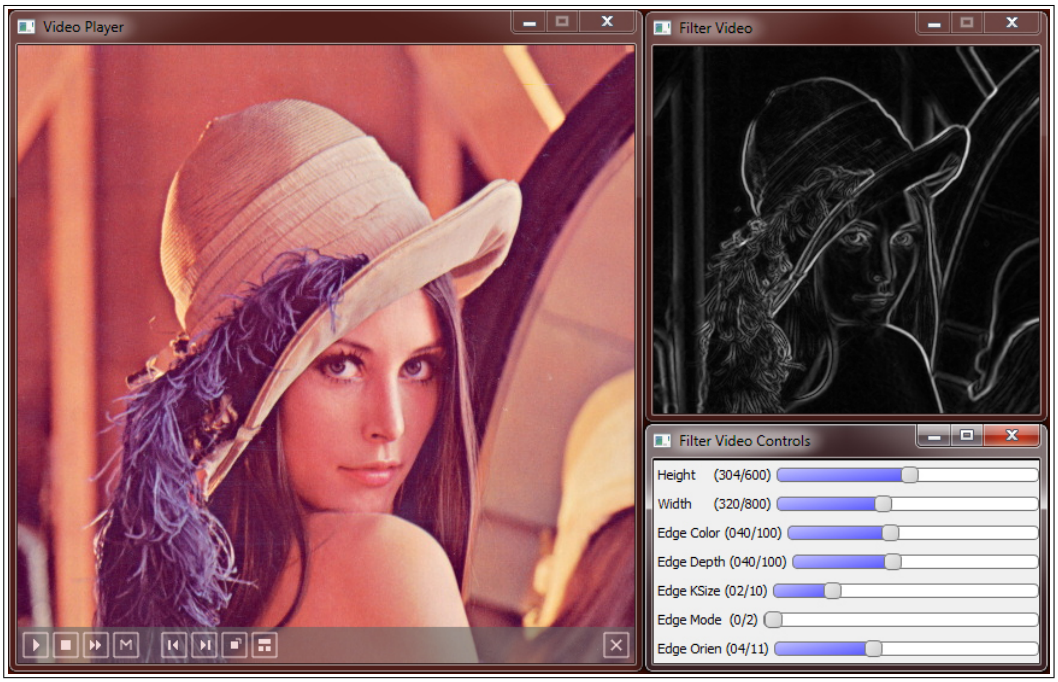


Figure 3.1: *The graphical user interface at work. The left window shows the video player with its controls at the bottom. The top right window shows the processed image, in this case a resize filter and an edge detection filter was applied. The bottom right window shows the registered parameters which can be changed on the fly.*

The implemented video player does not only provide a graphical user interface for the general video player controls and selecting the different video modes, but also for navigating through the different filters in the chain, detaching filter outputs to separate windows for better observation and adjusting registered parameters, as seen in fig. 3.1.

3.2 Algorithm and Modules

The pursued framework consists of different modules, each of them representing one step in the segmentation algorithm of the proposed approach:

- **Fixation Point Selection** implements the filter providing the selected fixation point.
- **Color Space Transformation** provides various filters for converting to different color spaces.
- **Edge Detection** contains all filters for detecting edges in a colored image.
- **(Log-)Polar Space Transformation** provides the transformation to (log-)polar space and back to Cartesian space.
- **Graph Cut** implements the graph cut used in [Mishra et al., 2009] for retrieving a rough object mask.
- **Grab Cut** implements the grab cut algorithm by using the results of the graph cut module as an input mask.
- **Contour Detection** provides the filter for finding the contour of the selected object by using the mask of the grab cut module.

By concatenating the various filters of the modules to one single chain, it is possible to define the arrangement of the components which in the end results in the segmentation of the desired object. [Mishra et al., 2009] propose an algorithm with different substeps which are shown in fig. 3.2.

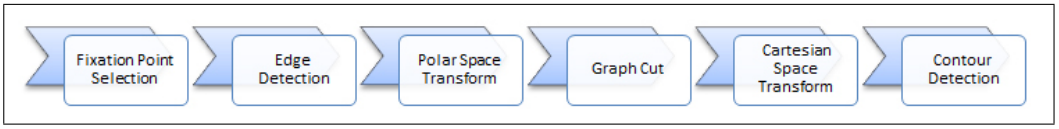


Figure 3.2: *The different substeps and their arrangement proposed by [Mishra et al., 2009].*

This presented framework adapts the algorithm and extends it with an additional grab cut filter after the graph cut filter. Furthermore, it permits to include one

of the different color space transformations before the edge detection, since the color similarity (or difference) is used to calculate the edge map. The expected impact of the different color spaces in general is discussed in section 3.4.

Another module, which relies on color information, is the grab cut filter. It employs color similarities to build up the Gaussian Mixture Models. Moreover, it uses the color- and depth-based edge map to calculate the weights of the n-links as covered in section 2.3.4. So, changing the color space might have an important effect on the quality of the result.

In contrast to the original algorithm, the edge detection is now done after the (log-)polar space transformation. The idea behind the proposed change lies in the nature of the human visual system. It is possible to exchange the (log-)polar transformation and the edge detection to achieve improved results of the edge map in regard to the graph cut algorithm[4]. The advantages are discussed in section 3.5.3.

Additionally, the simple polar space transformation used by [Mishra et al., 2009] is replaced by the log-polar space transformation[5]. The basic difference is presented in 2.2.1, whereas section 3.5.2 discusses the advantages and disadvantages regarding the edge detection.

The segmentation algorithm can now be defined as shown in fig. 3.3.

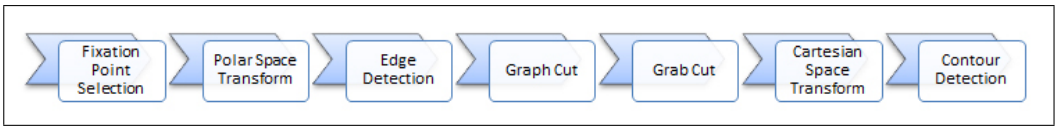


Figure 3.3: *The different substeps and their arrangement proposed in this thesis.*

[4]Due to the modular design of the filters, swapping the sequence of both filters require no additional programming. Therefore, both algorithms are supported.

[5]The configuration can actually be changed at runtime.

3.3 Fixation Point Selection

The fixation point selection module provides the information about the point of interest, more precisely the fixation point on the object. Since finding an adequate fixation strategy lies beyond the scope of this thesis, a simple user-dependent selection method is implemented. Being part of the filter chain, the user can select a point of interest by simply clicking on it with the mouse. As it is possible to change the location of a selected point during runtime, the evaluation of different object locations can be accomplished much easier.

An important issue regarding this module is the interchangeability with other implementations of fixation strategies. The interface provides a simple function to retrieve one or several points of interest. This makes it easy to replace this strategy with simple preselected static points or to use more sophisticated approaches like calculating salient maps or to even use hand gestures retrieved from the image beforehand.

As mentioned before, it is also possible to select multiple points of interest and thus simulate the human vision system, which alternates between saccades and fixation points. The segmentation process is then carried out for each point separately. As it does not affect the result of each independently selected object, this implementation disregards the use of multiple fixation points for the sake of computational performance.

3.4 Color Space Transformation

The aim of transforming the image into another color space is to find an appropriate representation of the perceived colors which allows to compute the similarity between them. There exist many approaches to express colors and their relations, like *RGB*, *Lab* or *HSV*, just to mention a few [Fairchild, 2005]. They all have their advantages and disadvantages depending on their use [Cheng et al., 2001].

This module implements the most common transformations used in computer vision: *RGB*, *YCbCr*, *LAB* and *HSV*. Additionally, it allows to define own hybrid color spaces by combining channels from different transformations, e.g. the lightness channel L from *Lab*, the hue H channel from *HSV* and the green channel G from *RGB*, resulting in an artificial color space *LHG*.

The color space transformation is applied before the edge detection and the grab cut module. In both cases, the color information is used in order to calculate the gradient between neighboring pixels. Additionally, the grab cut algorithm deploys color models which are based on this information to separate different color clusters. Therefore, the selection of the 'right' color representation has to take the use of the application into account.

The requirements for the color space in this framework are as follows:

- **Perceptual consistency:** The color space should represent the color similarity based on the human perception. As color is a perceptual impression, the distance calculations should take this fact into account instead of just comparing different wavelengths. A segmentation based on non-perceptual color spaces might give correct results, but cannot be comprehended by a human being.

- **Linear independence:** Each channel of the color space should represent a perceptually independent property of the color like lightness, hue and saturation. This allows to compare colors by analyzing each channel separately or by calculating the Euclidean distance between the color vectors. If the

channels are correlated, a modification in one of the properties will cause a change in all channels. This might lead to wrong results when comparing the color vectors by a distance measure. Often it is sufficient to separate the lightness and the chromaticity values, as these are the major indicators for different objects.

- **Robustness to noise:** The conversion of noisy images to a certain color space can introduce image artifacts, because the noise signal gets amplified. These color spaces should be avoided, as it can introduce sever errors when computing the color gradients.

The following sections discuss the different color spaces in regard to the requirements, followed by a comparison based on a *k-means* algorithm.

3.4.1 Standard Color Spaces

The color spaces used in this module can be divided into three groups.
The first group is the direct approach to represent color as a triplet of color components. These are the traditional color spaces like *RGB* and *CMY*. The CMY color space can be neglected, as it is a subtractive color space which is only used for printing.

In the second group are the color spaces which separate the brightness property of a color into its own channel. The hue and the saturation are given implicitly by two color components. Implemented examples are the *LAB* or the *YCbCr* color space.

The last group contains the circular color spaces which describe colors the most natural way by representing each perceptual property in its own channel. This module implements the *HSV* color space. In the following, the three groups are compared in regard to the given requirements.

The *RGB* color space is the most used color space in computer applications. Often, images, videos and camera streams are given in this color format. Therefore, applying an explicit conversion to *RGB* space is not necessary. This has the major advantage that the existing noise in the image does not get amplified, thus the effect on the color similarity measurement is reasonable. The disadvantage of this color space is the high correlation between the channels and the insufficient inclusion of the human perception [Littmann and Ritter, 1997]. Nonetheless, the *RGB* color space is a computationally convenient way of color representation. Fig. 3.4 shows a test image in *RGB* space.

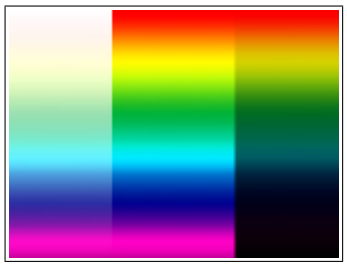
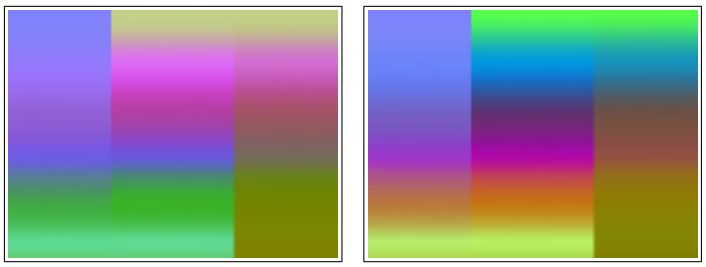


Figure 3.4: *A test image in RGB color space.*



(a) **(b)**

Figure 3.5: *A test image in the similar* ***(a)*** *Lab and* ***(b)*** *YCbCr color space.*

The second group, the *LAB* and the *YCbCr* space, are the color spaces which are widely used in computer vision. On the one hand, they decouple the lightness value from the color information, therefore providing a certain degree of independence, and on the other hand, they take the human perception into account [Cheng et al., 2001]. The *Lab* space is derived from the *CIE 1931 XYZ*[6] color space and is especially designed to match the human perception. The *YCbCr* space, in contrast, is an approximation of the *Lab* space, originally developed as a *RGB* encoding method for data compression. Both color spaces are additionally fairly robust to the exposure of noise, whereas the Lab color space transformation

[6]It is one of the first mathematically defined perceptual color spaces by the *International Commission on Illumination*. The color space was derived from empirical experiments [Wright, 1929, Guild, 1932].

tends to generate a little bit more artifacts in dark areas. Fig. 3.5 shows the *RGB* test image converted into *Lab* and *YCbCr* space.

The last group covers the circular color spaces like the implemented *HSV* color space as seen in fig. 3.6. It is the most intuitive color space, as the different perceptual properties are transferred one to one to the image channels. The *HSV* color space separates colors very well, but does not allow an easy distance calculation, since the values are arranged in a circle and the distance depends on the position of the colors on it. Calculating the distance by the color vectors would lead to wrong results, as the color models would cluster the wrong colors. E.g. the distance between the hue values 360° and 0° is calculated as the maximum distance, even though in reality they have the same hue. Another major drawback is the sensibility to noise. The transformation of *RGB* image exposed to noise causes many and strong artifacts, especially in dark areas of the image, due to the singularity at the center of the color model [Cheng et al., 2001].

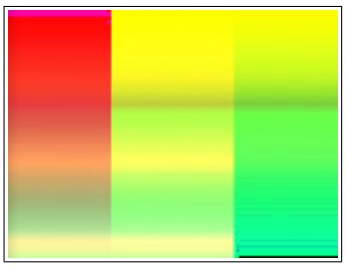


Figure 3.6: *A test image in HSV color space.*

3.4.2 Hybrid Color Spaces

Besides using the common color spaces for image segmentation, some approaches take the advantage of different color spaces by combining single channels of them [Boukala et al., 2003,Colantoni, 2004,Cheng et al., 2001]. The idea behind hybrid color spaces is to use the most discriminant channels such that the combination will maximize the separation between different colors. Even though, they might include color channels of perceptual color spaces, the hybrid space cannot be described in regard to perception, since it is a completely artificial color space.

The major challenge in the hybrid approach is to find the optimal color channels. As the optimal choice often depends on the image content, [Vandenbroucke et al., 2003] proposes an image segmentation method where the hybrid color space is automatically adapted to the analyzed image. However, due to the computational complexity, this algorithm cannot be implemented in this framework, as the selection of the color space is only an intermediate step in the whole algorithm. Hence, the selection of the hybrid has to be done manually. This framework provides the possibility to easily select different channels from various color spaces. Fig. 3.7 shows the original test image converted into the artificial *LHG* color space which is implemented in this module.

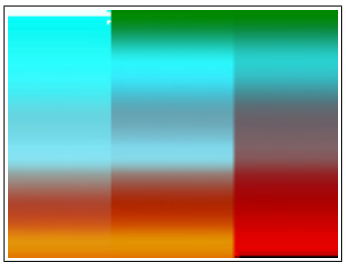


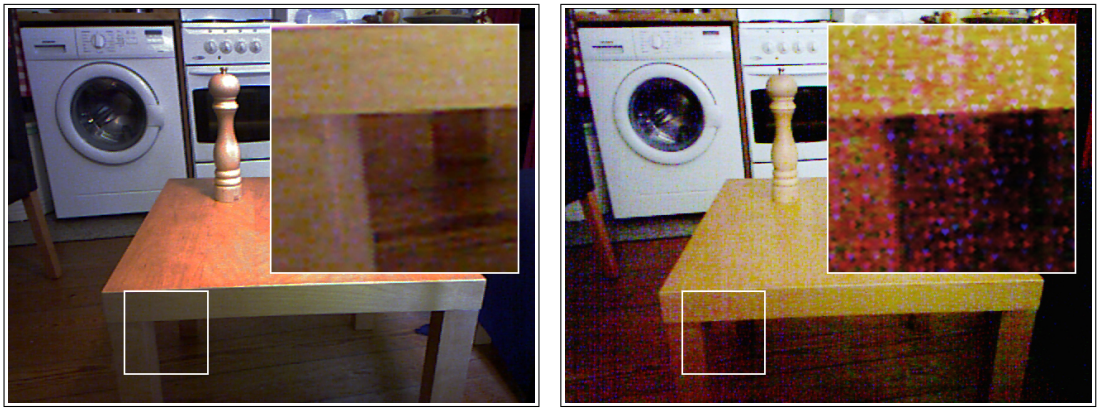
Figure 3.7: *A test image in the artificial LHG color space.*

3.4.3 Microsoft Kinect Camera

An important aspect when choosing the color space is the use of the framework for a robotic system. Thus, the input images used for the segmentation process are retrieved from a camera device, in this case the *Microsoft Kinect* camera. It has major disadvantages regarding the color image: As the camera is normally used as an input device for the *XBOX* game console, the quality of the color information is a minor issue. Therefore, the device is not reliant on high-quality images and is only equipped with a low-resolution camera (VGA) where the image sensor is a coarse Bayer color filter[7].

On the one hand, this leads to a considerable sensibility to noise, especially when used in a dark environment. On the other hand, additional image artifacts are introduced when decoding the Bayer color space to the common *RGB* space. This is caused by the Bayer filter as each pixel is represented by two green, one red and one blue filter, arranged in a 2×2 array. The transformation into *RGB* space

[7]See http://openkinect.org for more details.



(a) **(b)**

Figure 3.8: ***(a)*** *Contrast-enhanced image retrieved from the Kinect camera in a bright environment.* ***(b)*** *Contrast-enhanced image retrieved from the Kinect camera in a dark environment. Many red and blue artifacts appear in the image.*

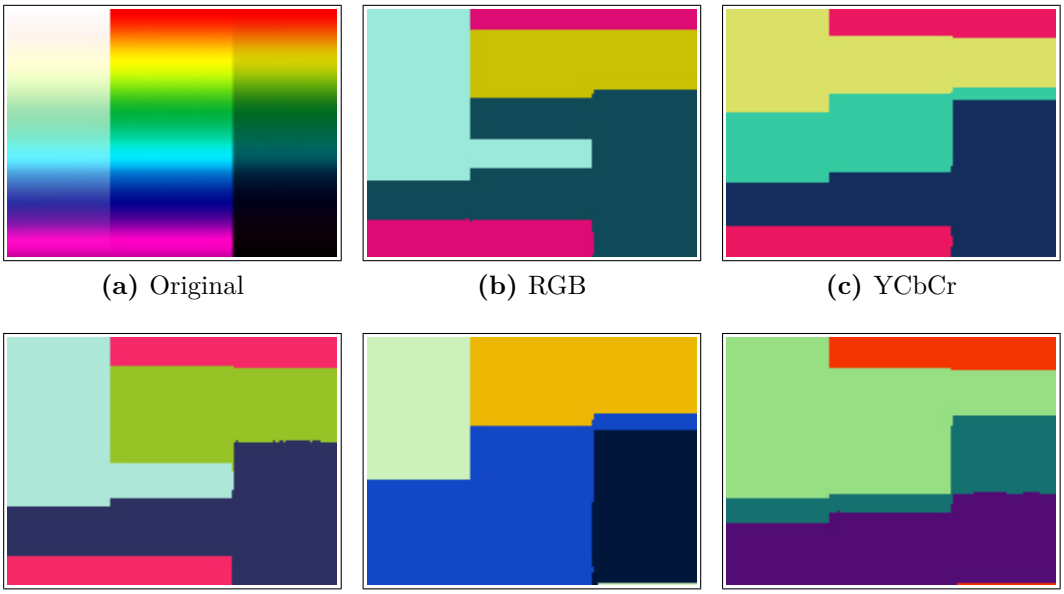
thereby generates in particular red and blue image artifacts as seen in fig. 3.8b. The additional transformation to another color space will amplify these artifacts.

3.4.4 Comparison of the Color Spaces

In general, there is no optimal color space, as it often depends on the scene, the input color space and the quality of the image. Nevertheless, it is possible to compare the different color spaces in regard to the requirements of this application. Therefore, a test image is evaluated by clustering the colors in each color space using a simple k-means algorithm. The same procedure is repeated using the same test image exposed to 10% Gaussian noise. These results are then compared by means of the requirements.

Fig. 3.9 shows the clustering of the test image in different color spaces. It shows which colors in the color spaces have the highest similarity. Even though all the color spaces result in different clusterings, it is hard to tell which is the optimal segmentation, as each of them is correct depending on the context. In the context of object segmentation, the goal is to distinguish objects based on the color. As mentioned in section 3.4, the most important color characteristics of different objects are the lightness and the chromaticity. Therefore, the best result would be a segmentation of the test image where the hue values from top to bottom

are separated. This results in horizontal clusters except for the white or black area which should be clusters on their own. The best ratio between separating chromaticity and lightness are provided by *YCbCr*, *Lab* and *RGB* color spaces. The *HSV* color space is too sensible to lightness, therefore prefers the clustering of dark and light areas. The artificial *LHG* is very tolerant in brighter areas and includes many hues, whereas in dark areas the separation of different hues increases.



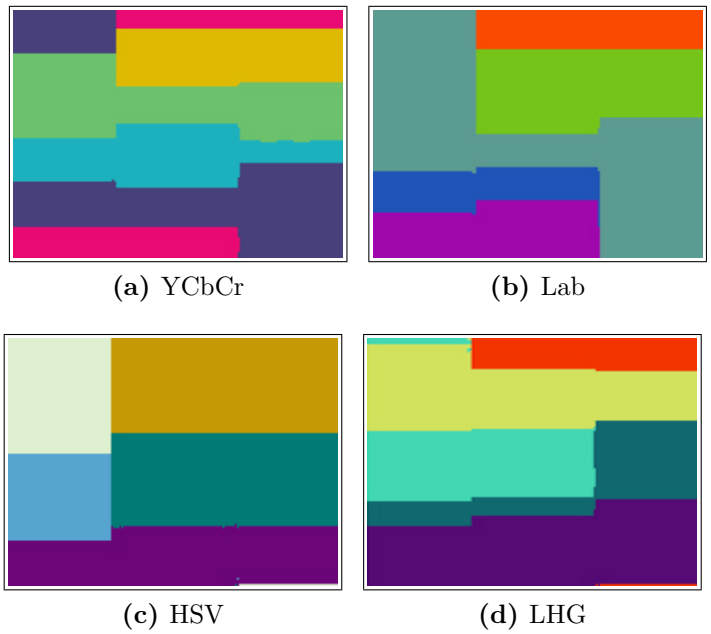
(a) Original **(b)** RGB **(c)** YCbCr

(d) Lab **(e)** HSV **(f)** LHG

Figure 3.9: *The comparison of the k-means clustering with four components in different color spaces.*

All color spaces except the *RGB* space try to separate the lightness from the chromaticity information. This can be used in order to split the identification of an object's boundary into two distinct steps: One for the lightness and one for the chromaticity. The similarity of lightness values is trivial, as it can be reduced to simply comparing gray scale values. However, the comparison of the 'pure' colors is still a perceptual problem: Therefore, the clustering of the test image is repeated independent of the lightness property as shown in fig. 3.10

The results of the clustering show the linear dependency of the chromaticity and the lightness. In the ideal case, the result would be horizontal rectangular regions covering the whole width of the image. All color spaces give good results, especially the *YCbCr* space, which comes closest to the optimum. The *HSV* space makes the best separation of hue values, but is still too sensible for bright colors. The *LAB* and *LHG* space provide results in between, whereas the *LAB* space is closely related to the *YCbCr* space and the *LGH* performs a little bit better than the *HSV* space.

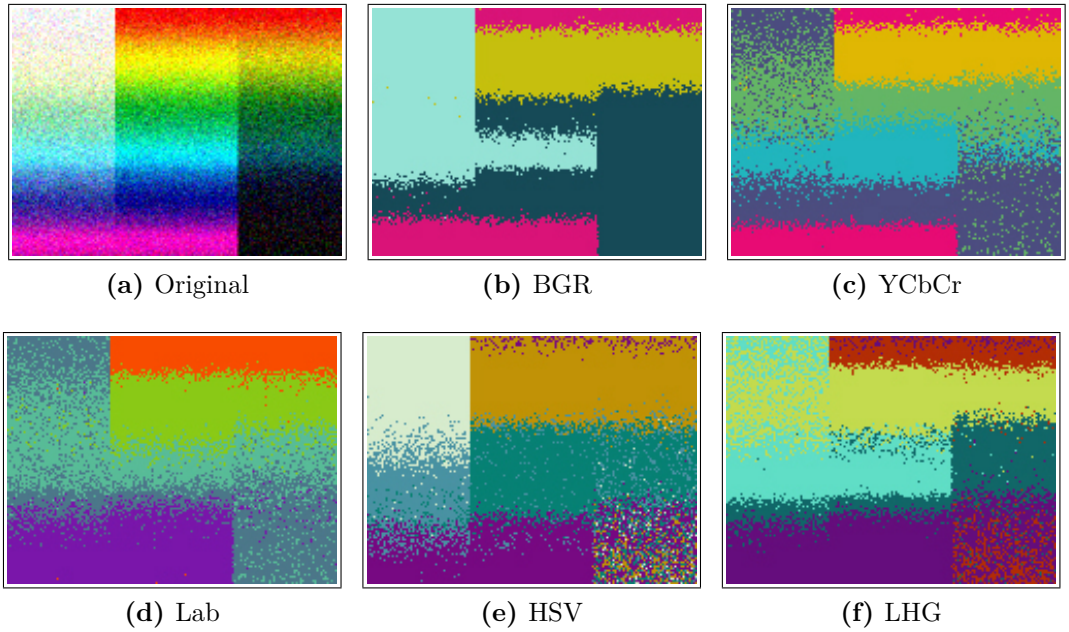


(a) YCbCr **(b)** Lab

(c) HSV **(d)** LHG

Figure 3.10: *The comparison of the k-means clustering (k=5) of chromaticity in different color spaces.*

The last important criterion, which has to be tested, is the resistance to noise. Therefore, the experiment is repeated with the same test image exposed to 10% of Gaussian noise. The results are shown in fig. 3.10. As expected, the *RGB* clustering is not much affected by the noise. The circular color spaces (*HSV*, *LHG*), in contrast, generate strong artifacts in dark regions, especially when converting to *HSV* color space, the dark region contains pixels from all other clusters as well. The *Lab* and the *YCbCr* color space respond moderately to the exposure of noise.



(a) Original **(b)** BGR **(c)** YCbCr

(d) Lab **(e)** HSV **(f)** LHG

Figure 3.11: *The comparison of the k-means clustering (k=5) of the test image exposed to 10% Gaussian noise.*

The results of these tests show that the *HSV* space should be avoided, as well as any circular color space, even though the *LHG* space performs better. The most preferable properties offers the *YCbCr* color space closely followed by the *Lab* space. The *RGB* color space should not be neglected due to its almost optimal response to noise. This can come in handy when using the *Microsoft Kinect* camera system.

3.5 (Log-)Polar Space Transformation

The polar space transformation is the crucial point of the approach from [Mishra et al., 2009] for solving the 'shortcut' problem of the graph cut algorithm, as covered in section 3.8.2. Even though the transformation is theoretically a bijective function, when applying it to all coordinates of an image, a discretization has to be made which results in an interpolation between the pixels. In general, a bilinear interpolation is used when the value of a subpixel has to be calculated. Instead of using the four neighbors to interpolate such a subpixel, [Mishra et al., 2009] propose a continuous 2D function by placing 2D Gaussian kernels on each pixel.

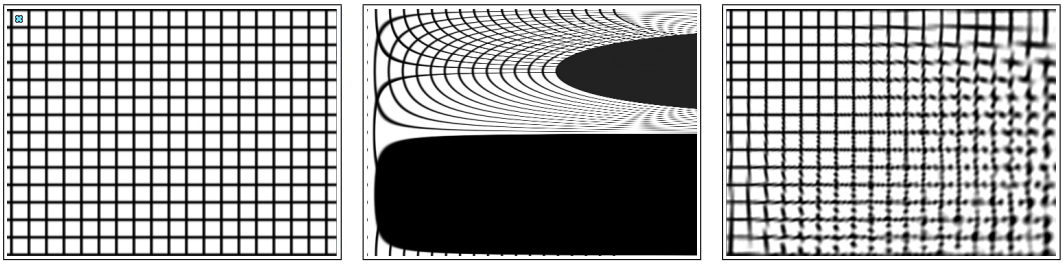
They use this approach to keep the intensity values of the pixels steady and to vary the influence of the intensities depending on the pixels' distance to the pole. The idea behind this is that pixels more distant to the pole appear smaller than pixels close-by. However, this calculation and optimization in order to get better results costs additional computational time.

3.5.1 (Log-)Polar Coordinate Mapping

The proposed implementation of this thesis reduces the polar space transformation to a simple mapping function by computing the coordinates' mapping beforehand. This means the effort of calculating will have to be done only once if the fixation point does not change. So using the same mapping for different images (e.g. the original scene and an edge map) can speed up the segmentation process. It can also be easily applied to videos or camera scenes, where the fixation point does not change. However, since the content of the image is not known or can change, the interpolation of the pixels has to be done for each image separately. In this case, the traditional bilinear interpolation is used, which leads to a good compromise between performance and quality. The mapping to the polar space results in a transformed image where the vertical axis represents the angle

$\theta \in [0°, 359°]$ increasing from top to bottom while the horizontal axis represents the radius $0 \leq r \leq r_{max}$ with r_{max} denoting the image diagonal.

For the transformation from polar space back to Cartesian space, the same algorithm is used. The major problem of this transformation is the interpolation due to the discretization. Artifacts resulting from this retransformation are much more visible than those from the polar space transformation. The further away from the fixation point, the more pixels have to be interpolated. Given that the object of interest is always located close to the fixation point, this fact is, in this context, negligible. Therefore and because of performance reasons, implementing a sub-sampling algorithm is abandoned. Fig. 3.12 shows a polar space mapping and the retransformation of a mesh with a given pole at the top-left corner of the image. After the retransformation, the image contains mapping artifacts which grow with the distance to the pole. Note, that the mesh close to the pole suffers much less from this effect and still has the same quality as the original mesh.



(a) A mesh in original Cartesian space.

(b) The mesh in polar space.

(c) The mesh after the remapping to Cartesian space.

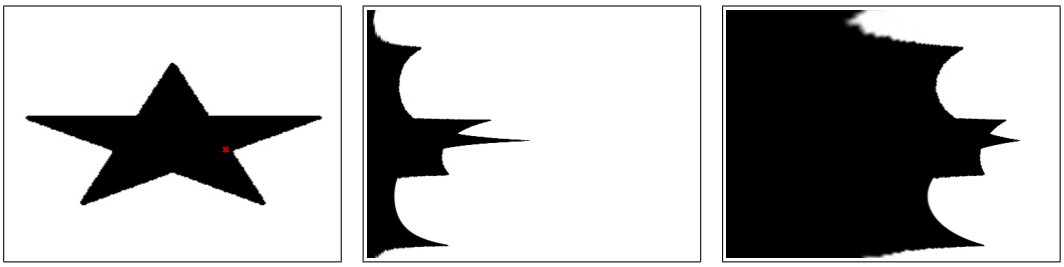
Figure 3.12: *The effect of the polar space mapping and the remapping to Cartesian space on a mesh.*

Another drawback of this transformation, resulting from the interpolation and the transformation map, is the occurrence of an artifact while interpolating $\theta_i = 259°$ with pixels from the next row $\theta_{i+1} = 360°$ where the pixel values are not defined. Hence a border handling function has to be implemented what will be covered in subsection 3.5.4.

3.5.2 Simple Polar Space vs. Log-Polar Space

Another important point to consider is the choice between the simple polar space and the log-polar space. This framework actually implements both approaches. Their expected advantages and disadvantages will be discussed in the following.

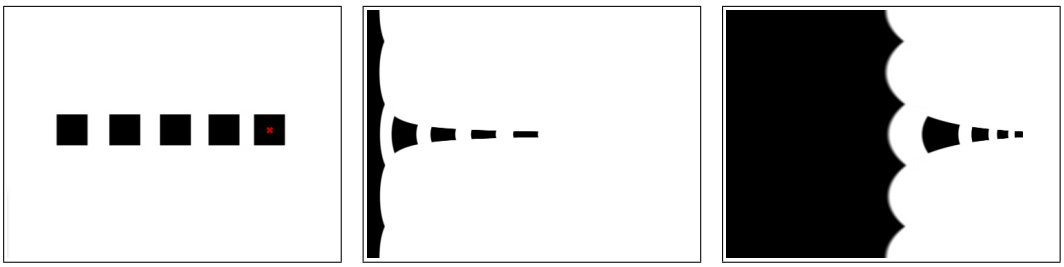
When using the polar space, the radius increases constantly (from left to right in the transformed image), whereas the distance between two polar angles in respect to the radius increases exponentially, thus causing objects to deform, more precisely, to elongate along the horizontal axis. This might not be a problem for objects further away from the pole, if the edge detection has taken place before, since they are not of interest anyways, however if the object of interest has already have elongated regions, e.g. a starfish, these regions would be stretched even more. In the later segmentation process, when using the graph cut and grab cut algorithm, this can lead to cut-off regions due to the 'shortcut' problem, as discussed in section 3.8.2. Fig. 3.13 shows the difference of an star transformed into polar space and log-polar space.



(a) A star in Cartesian space. **(b)** The star in simple polar space. **(c)** The star in log-polar space.

Figure 3.13: *The difference between the simple polar space and the log-polar space of a transformed star. Already elongated regions get stretched even more in polar space.*

Note that in polar space the spikes of the star get much more elongated with the distance to the pole. The log-polar space, in contrast, tries to avoid the deformation of objects along the radial axis. By using the logarithmic scale for the radius, regions close to the pole are expanded while regions further away get contracted. As the distance between two polar angles in respect to the radius grows exponentially, the logarithm evens out the exponential growth such that quadratic regions stay almost quadratic, as fig. 3.14 shows.



(a) Different sized squares in Cartesian space. **(b)** The squares in simple polar space. **(c)** The squares in log-polar space.

Figure 3.14: *The difference between the simple polar space and the log-polar space of transformed squares. The log-polar keeps the aspect ratio of objects.*

Another effect resulting from the log-polar transformation is the expansion of the (probable) object region. On the one hand, the expanded region can be used to build better color models for the later grab cut algorithm, since more pixels are available, on the other hand, it might be a considerable disadvantage for identifying small objects, because regions close to the pole are heavily interpolated. Thus, existing edges can get blurred in such a way that the following graph cut algorithm will not recognize them as actual edges.

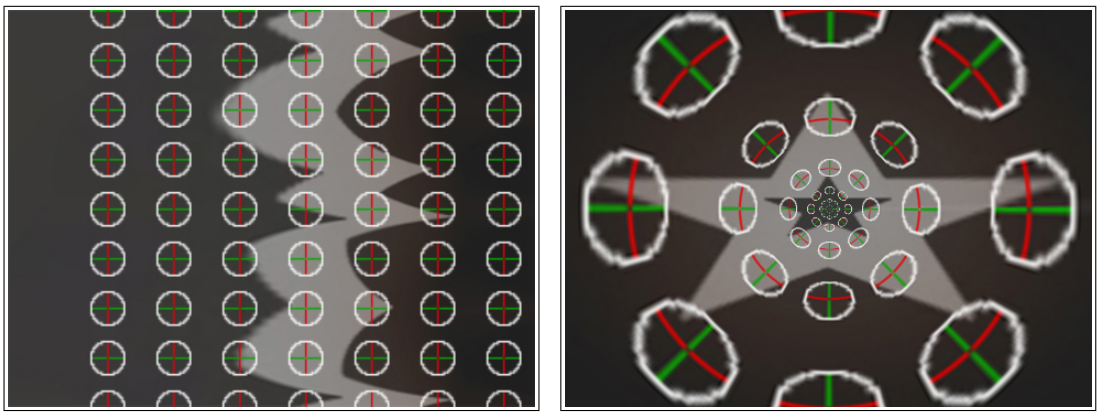
(a) A rainbow circle in Cartesian space. **(b)** The rainbow circle in simple polar space. **(c)** The rainbow circle in log-polar space.

Figure 3.15: *The difference between the simple polar space and the log-polar space of transformed squares. The log-polar increases the object region and therefore allows to build more precise color models for the grab cut algorithm.*

Fig. 3.15 shows an image of a circle filled with layers of different colors. The log-polar transformation shows that the region close to the pole is strongly expanded. This region can be used to create a more precise color model during the grab cut algorithm. Especially in textured regions, this approach can improve the results of the grab cut algorithm enormously.

3.5.3 Polar Space and Edge Detection

Depending on the sequential arrangement of the filter chain, the edge detection can also take place after the polar transformation, as mentioned in section 3.2. When using edge detection filters with a large kernel size, the choice of using the simple polar space or log-polar space transformation can have a huge impact on the form of the kernel seen from Cartesian space's point of view. Especially when using the multi-oriented edge filter, which uses a disc kernel, the information used to retrieve the edge can vary immensely. Fig. 3.16a shows a log-polar transformed star with overlaid oriented edge kernels.



(a) A star in log-polar space with an overlay of oriented kernels.

(b) The star and the kernels retransformed into Cartesian space.

Figure 3.16: *The applied edge detection kernel in log-polar space equals the edge detection with an increasing kernel in Cartesian space. The kernel is oriented along rays emanating from the pole.*

Convolving an image with a disc kernel in (log-)polar space can be transferred back to a convolution process in Cartesian space. Therefore, the kernel has to be retransformed from polar space (where it is a disc). Since the polar space deforms objects depending on their distance to the pole, the kernel form varies accordingly in Cartesian space. Additionally, the kernel is not moved along the x- and y-axis, but along a ray emanating from the fixation point with an increasing radius. Using the simple polar space implies that a round kernel will become more and more bean-shaped with increasing distance to the fixation point due to the reasons mentioned in subsection 3.5.2. When applying the disc kernel in log-polar space, the transformation of that kernel into Cartesian space results in

a circle again which grows with its distance to the fixation point. Close to the fixation point, the kernel is almost reduced to a single pixel. Fig. 3.16b shows the accordingly retransformed edge kernels in Cartesian space.

Taking the human visual system into consideration once again, it becomes clear that using the edge detection module after transformation into (log-)polar space is an suitable approach. When the human eye focuses on an object, the macula – the area on the retina with the highest resolution power – is straight in the center, whereas the areas further away begin to blur. The use of a kernel size increasing proportional to the distance from the fixation point can be seen analogously. As the probability of a region belonging to the fixated object decreases with the distant to the pole, it has the advantages that the edge detection uses a larger kernel for these less important regions which results in less detailed edges.

3.5.4 Border Handling

The implementation of the transformation into polar space is done by remapping the image coordinates. This comes with the cost of additional border handling. Besides generating artifacts during the mapping, the border handling is also important for the edge detection after the polar transformation. Due to the representation in polar space, a large region of the image is now undefined, as there are no corresponding coordinates in Cartesian space. The fastest way to deal with it is to leave those parts blank or fill them with a constant color (in case it is not an edge map). When the unmapped pixels are used for interpolation, this can result in unwanted artifacts. Another problem occurs when convolving the polar transformed image with the edge detection filter: If the color of the background has a constant value, the transformed image frame is detected as a strong edge, an effect which might be obstructive for the following graph cut algorithm.

In the first case, when transforming the image back to Cartesian space, the best solution is to replace the second interpolation pixel from row $\theta_i = 359°$ to $\theta_{i+1} = 0°$, that is using the border wrapping method to connect first row of the image

with the last one. In the second case, a different border handling is required. Filling the empty parts of the image with the same image again would result in additionally detected edges. The easiest way to solve this problem is to fill pixels which belong to coordinates outside of the original image with the same color as the corresponding ones on the border. This can be done by cropping the x- and y-coordinate to the width and height of the image boundary respectively.

3.6 Edge Detection

The edge detection module is one of the most important modules, since it serves to produce the input mask for the grab cut algorithm used later on. Creating a probability boundary map based on sloppy edges or extracting too many irrelevant or too few edges can lead to a malfunction of the whole working process. In this case, the graph cut algorithm would select a too small or too large area, so that after learning the colors for the foreground color model, it would not represent the true colors of the object. Then again, the edge detection is also the computationally most complex module (see chapter 3) with the widest scope of optimization for achieving a real-time system. Therefore, a good balance between quality and performance is required.

[Mishra et al., 2009] propose a complex edge detection algorithm, which is based on the probabilistic boundary detector of the Berkeley University [Martin et al., 2004]. The researchers implemented edge detectors for different visual cues like brightness, color and texture. The individual results are optimized and combined to a single probabilistic boundary map (which is actually an edge map). The detailed description of the implementation can be found in [Martin et al., 2004].

Even though this approach gives very good results of the edge map, it implies a lot of steps and optimizations with high computational costs which are disproportional to the received quality gain. Most of the conducted optimizations regarding the location criterion of the edge, for example the non-maximum suppression, can be neglected, since the graph cut algorithm using this edge map is not necessarily reliant on thin edges. As long as the edge maximum is positioned at the right location, the graph cut algorithm will find the 'right' edge. This actually applies to spurious edges as well, since they will be ignored, as long as they are not too numerous.

In order to optimize the edge detection process, the criteria have to be defined anew for the purpose of using the detector with the graph cut algorithm. In the following, the criteria and the respective solution approaches are discussed:

- The most important criterion is the solid detection of all strong edges. Strong edges are defined as a rapid change of brightness or color. When using colored images, the edge detection has to be applied on each channel of the image. It also is important to choose an appropriate color space as described in 3.4. Most of the simple edge detectors like Prewitt or Sobel can identify such strong edges.

- The robustness to noise is important, since the implemented algorithm is supposed to work with a robotic system as well. Typically, the noise reduction is done by a smoothing filter like the Box filter or the Gaussian filter. The smoothing highly depends on the kernel size.

- Textures tend to produce strong edges as well. This problem of textures can be difficult to handle, since it depends on the viewer's interest. That means, if there is an area with a large texture, like a closeup on a chess board, it will be obviously not clear if the viewer is interested in a single square or the whole board. Small textures can be sufficiently smoothed by a larger kernel size. Additionally, the grab cut algorithm in the end can counteract not perfectly separated texture regions by using the color models. Another strategy to handle textures, especially on a large scale, is to additionally use stereoscopic cues like a disparity map. The disparity map only creates edges where the depth values of regions change abruptly, therefore it will not create edges on a single flat surface, even if they have textures on it.

- Spurious edge fragments are often generated by simple edge detection filters. The edges do not have a strong intensity and they tend to be a short contour. Therefore, they are not overly relevant for the graph cut algorithm, since it will find 'shortcuts' around those regions. But if there are too many spurious edges, it will influence the graph cut algorithm in a negative way.

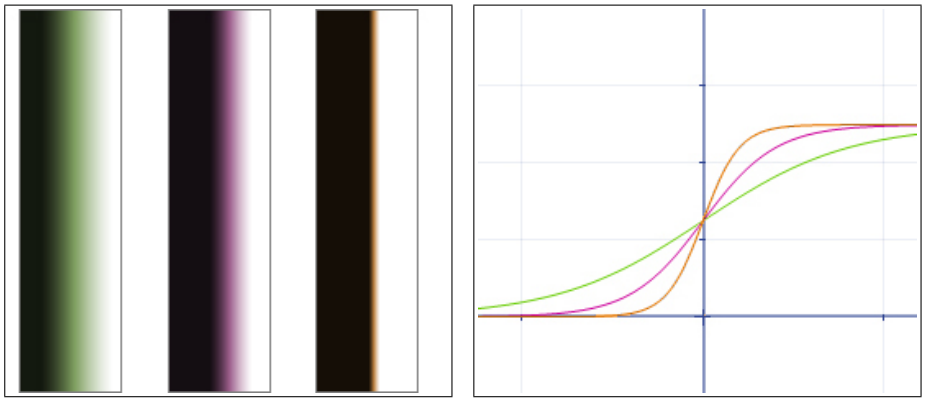
- As the goal is to find an object's contour, the localization of edges is quite important. But, as mentioned before, it is only important that the highest intensity of the edge is at the right place. The graph cut algorithm is insensible for low-intensity edges. Furthermore, the localization of the edges is refined with the grab cut algorithm, as the color models are used in order to find the exact edges.

- Multiple responses of an edge can also be neglected when using a larger kernel size, as multiple responses tend to appear close to each other. As long as a non-maximum suppression is not applied, the smoothed edges will 'grow' together. Even if there are multiple responses of the same intensity, the graph cut algorithm will take the 'shorter' one. If this is the 'wrong' edge, it will be handled the same way as a 'wrong' located edge, subsequently refined by the grab cut algorithm.

Taking these criteria into account and summarizing the proposed solutions, three major characteristics can be derived for the edge detector: Firstly, the edge detection should take at least the two visual cues lightness and color into account and combine them to generate the edge map. Secondly, the edge detection has not to be 'optimal' like in other approaches, since the graph cut and grab cut algorithms are capable of dealing with minor errors. Finally, the use of a larger kernel for smoothing is preferred, as it can counteract some of the effects listed above. At the same time a larger kernel means that valid high frequency edges are neglected, therefore it might be difficult to find the 'right' size. Ideally, it will be the best if the kernel size is variable, i.e. edges near the object of interest are detected with a smaller kernel in order to find more details, whereas more distant regions can be united by using a larger kernel size. An approach for using a variable kernel size can be found in section 3.5.3.

3.6.1 Kernel Design

Taking these aspects into account, this edge detection algorithm implements a new filter which is a simplified form of the derivative of Gaussian kernel as used in the Canny edge detector. On the one hand, there are the simple kernel operators like Prewitt and Sobel. Both of them are based on the gradient of the image intensities, whereas the Sobel kernel also includes a smoothing term. However, both of them do not provide different kernel scales, especially for large kernel sizes. On the other hand, there is the Canny algorithm. It implements the derivative of Gaussian as the smoothing and edge detection filter by combining the Gaussian function and a first-order derivative filter, e.g. the Prewitt operator. Additionally, the Canny algorithm uses the non-maximum suppression and hysteresis thresholding for refining the edges. These processes are obsolete, as discussed earlier.



(a) Three differently smooth edges. **(b)** The accordant step functions.

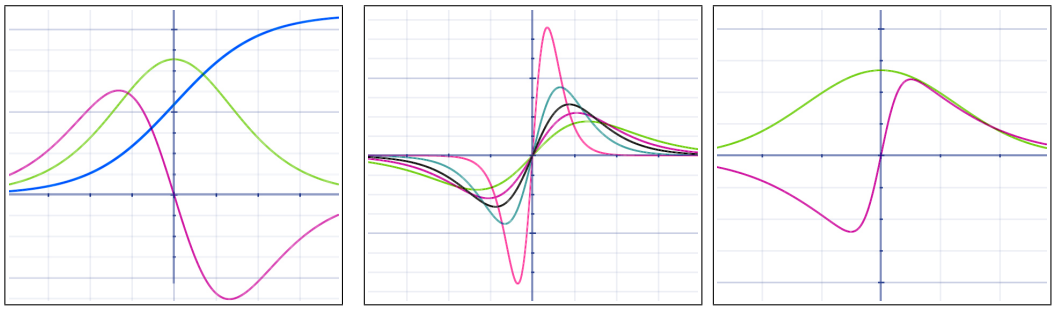
Figure 3.17: *Three examples of smooth image edges approximated by a logistic function.*

The derivation of the new kernel design is based on the assumption that an edge in an image can be expressed with the Heaviside's unit step function[8]. It is a discontinuous function where the function value for negative arguments is zero and one for positive arguments. For binary images this is obviously a valid assumption, if black pixels are denoted as zero and the white pixels as one. The step function thereby denotes the 'optimal' edge. However, in gray-scale images, the transition

[8]See [Bracewell, 1999] for more information about the unit step function.

between a black and a white pixel is often smooth. For a smooth approximation of the unit step function, the logistic function can be used [Bracewell, 1999]. The edge thereby becomes a continuous function which is shown in fig. 3.17.

Now, the first derivative of the edge function is a Gaussian function which can be interpreted as a smoothing kernel. The smoothing of the 'optimal' edge with this kernel results in exactly the given edge function. The second derivative of the edge function is the derivative of Gaussian which can be used as an edge detection kernel. Therefore, for each edge function exists exactly one 'optimal' derivation of Gaussian filter. Fig. 3.18a shows the derivatives of one edge function. Assuming that there are various edges in an image with differently smoothed edge functions, it is possible to use multiple derivation of Gaussian filters to detect the 'optimal' edges of each.



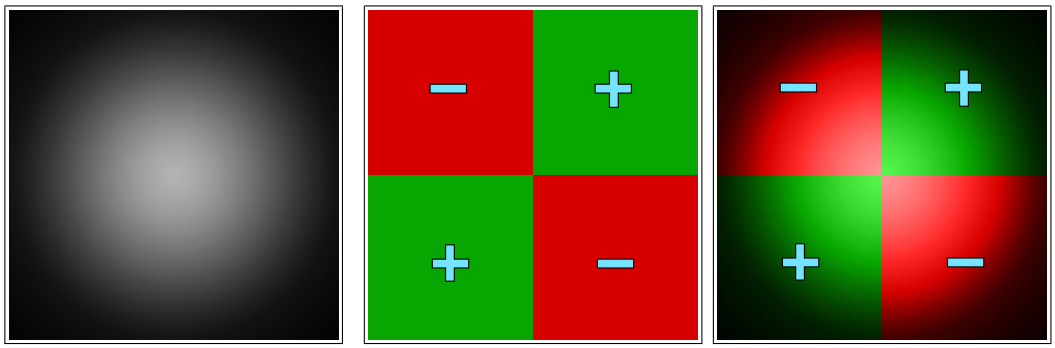
(a) The edge function (blue) and its derivatives (green, pink).

(b) Multiple DoG functions.

(c) Approximation of the average DoG function (pink) by a Gaussian function (green).

Figure 3.18: *The approximation of the averaged derivation of Gaussian functions by a Gaussian function.*

By selecting of a range of edge functions (fig. 3.18b) for edges which should be detected, a new average edge function is created which is similar to the original edge function, but the maximum and minimum are closer to the y-axis. This function can be approximated by a regular Gaussian function, as shown in fig. 3.18c. The only difference is that the Gaussian function returns positive values for negative arguments. However, this can be easily adapted by inverting the function for the negative arguments.



(a) The 2D-Gaussian kernel. **(b)** The inversion mask. **(c)** The approximate derivation of Gaussian kernel.

Figure 3.19: *The generation of the implemented edge kernel. Red denotes negative and green positive values.*

This derivation of the edge detection function is used to implement a new kernel design which is very accurate and can be computed very fast. Instead of computing the derivative of a Gaussian or convolving the Gaussian function with a gradient filter, the *OpenCV*-preimplemented Gaussian kernel is only multiplied with a mask which inverts the 'right' parts of the Gaussian in regard to the derivation of Gaussian filter. Fig. 3.19 shows the computation of the edge detection filter in two dimensions.

3.6.2 Disparity Edge Map

A completely different way to enhance the edge detector is the use of the *Microsoft Kinect* sensor. It allows to expand the edge detection not only by employing monocular cues, but also by using the information of stereoscopic vision. The *Kinect* device uses an infrared depth sensor to generate two depth maps of the visual scene which can be used to create a disparity map. Since this process is implemented directly on the *Kinect's* hardware, the performance is not constrained contrarily to the manually created disparity map used in the approach of [Mishra et al., 2009]. A major disadvantage of the created disparity map is the very coarse result. The infrared sensor generates shades behind occluded regions which is often the case near object boundaries. Furthermore, the object's edges

will not be traced exactly due to different reflectional behavior and noise. An example of a retrieved disparity map is shown in fig. 3.20.

Figure 3.20: *A disparity map retrieved from the Microsoft Kinect sensor.*

Another issue resulting from a separated color image and a disparity map is that both images have to be aligned in a way that objects in the disparity image are mapped to objects in the color image. Even though, this can effectively be done by the *Kinect* hardware, the image still has to be cropped, since the alignment creates image borders in the disparity map.

Nevertheless, the depth information can contain important information about the property of the object's surfaces. If there is an object with a strong textural boundary, the monocular edge detector will give a strong response, whereas the disparity map will be smooth. So, by extracting strong edges from the disparity map and combining them with the edges detected using the monocular cues, true boundary edges of the objects can be amplified.

3.6.3 Combining the Edge Maps

The final computation of the edge map is done by using monocular and stereoscopic cues. On the one hand, the proposed kernel is used to calculate the lightness and color gradient gradient map and on the other hand, the disparity map is also convolved with the same kernel to generate the depth edge map. Since this filter is an oriented filter like the Prewitt filter, multiple filter passes with different directions are applied. The maximum value over the orientations is then assigned to each edge map.

The difficulty, when combining the different edge maps, is how to choose the weighting for each map. The color and the lightness edge map contain similar edges, whereas the lightness edge map is slightly more robust to internal texture edges. Therefore, an almost average weighting should be selected with a tendency towards the lightness edge map.

However, the major problem arises from the merging of the color/lightness edge map and the depth edge map. On the one hand, the depth edge map produces strong edges for the 'true' object's boundary, i.e. the texturing of objects does not generate edges. This is the case, as long as there are not multiple objects with the same distance close to another. On the other hand, if two objects are very close such that they share a common boundary, the depth edge map cannot detect this edge, as the depth information are the same for both objects. Such a boundary is called a contact boundary. The problem is that such an contact boundary always exists between the object and the surface it is standing on.

Having said that, finding the optimal mixing parameter is a cumbersome task and often depends on the image contents. On the one hand, the depth edge map can be applied to remove most of the internal edges caused by textures. This implies a high weighting of the disparity edge map. On the other hand, the weighting should not be too strong, since the detection of the contact boundaries relies solely on the color and lightness information. Moreover, if the disparity map is of low-quality like the one of the *Microsoft Kinect* sensor, the computed depth edge of the objects are often dislocated due to noise, even though they tend to be very strong. In this case, the weighting of the color and lightness edge map has to be increased additionally to even out the errors of the depth edge map.

3.7 Graph Cut

The graph cut module is responsible for generating a first rough mask of the object's area. It is applied on the polar transformed edge map to separate the object from the background region. The binary output mask and the original color converted image are then used in the following grab cut algorithm to additionally incorporate the missing color information.

In order to apply the graph cut algorithm, the edge map has to be transformed into a graph. Therefore, every pixel is considered as a node of the graph. Like the pixels in the polar space, the nodes of a row in the graph represent the pixels located on a ray emanating from the fixation point at an angle equal to the row index. This means that nodes on the left-hand side of the graph are located closer to or within the object, whereas the right part contains nodes more distant from the object, thereby representing probable background pixels. The nodes are connected to all neighboring nodes by weighted links. The weights depend on the energy function used and the intensities of the edge map. In comparison to the approach of [Mishra et al., 2009], an 8-neighborhood is used instead of the simple 4-neighborhood. Even though this adds up to the time and space complexity, it reduces the metrication artifacts produced by a 4-neighborhood which means the retrieved contours will not get too 'blocky'.

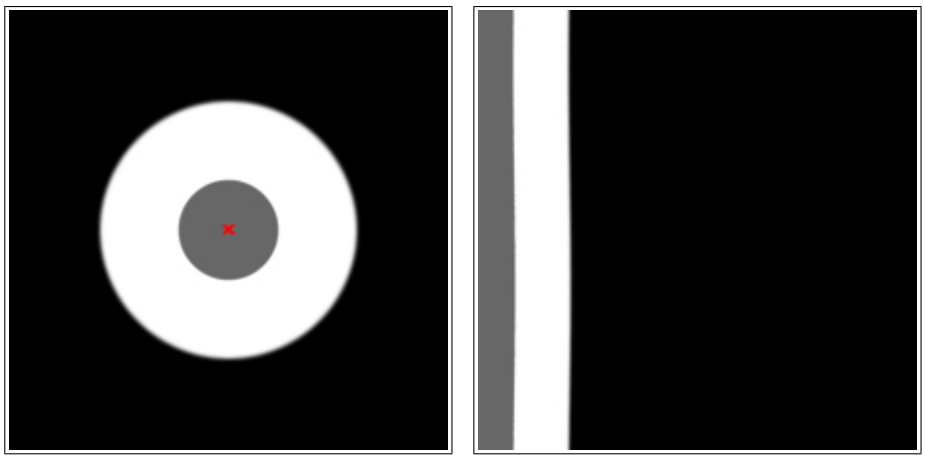
Another aspect which has to be considered when working with the grab cut algorithm in polar space is the border handling as described in section 3.5.4. Pixels at the top and bottom row are actually neighbors. Therefore, when mapping the image to polar space, the image is wrapped around itself, i.e. the image repeats itself by virtually 'copying' the bottom row above the top row and vice versa. If the wrapping is not carried out for the graph cut algorithm as well, the generated minimum cut could end at different radii at the top and bottom row respectively. This would result in a gap when the processed image is transformed back to Cartesian space, whereat the contour detection procedure would find a closed contour with a sudden angular notch in it. In order to avoid this

behavior, [Mishra et al., 2009] propose a simple method: Instead of replicating the image in the vertical direction, the nodes of the top row are also connected to the nodes of the bottom row. Hence, the cut is guaranteed to be 'smooth'.

Now, the goal of the graph cut algorithm is to find a labeling for each pixel of the edge map which results in a mask with the two labels 'foreground' or 'inside' the object and 'background' or 'outside' the object. This can be achieved by applying the min-cut/max-flow algorithm on the graph. It splits the edge image into a left part (inside) and a right part (outside) by minimizing the energy function used for initializing the weights between the nodes.

3.7.1 Graph Cut in Polar Space

The reason for using the polar space for applying the graph cut algorithm is the scale invariance of the polar space. This means that identical objects only differing in their size will have the same contour length. This actually addresses the common problem of graph cut-based approaches solved by a min-cut/max-flow algorithm, which tends to be biased towards small contours. Fig. 3.21b shows a disc object which consists of two circles. The big circle is the actual boundary whereas the inner circle is just an internal edge on the disc.



(a) **(b)**

Figure 3.21: ***(a)*** *A disc consisting of two circles with different intensities.* ***(b)*** *The corresponding disc in polar space with the pole in the center of the disc. The vertical axis represents the angle, the horizontal axis the radius.*

The radius of the circles are 26 and 70 pixels resulting in a perimeter of 164 and 440 pixels. The intensities of the circles ranging from 1.0 for white and 0.0 for black, are 0.4 and 0.7 respectively. The costs of tracing the contour for a graph cut algorithm in Cartesian space will be $98 \cong 164 \cdot (1 - 0.4)$ and $132 \cong 440 \cdot (1 - 0.7)$. The contour costs of the inner circle are clearly smaller than the outer one, thus it will become the optimal contour even though the edge between the outer circle and the background is much stronger.

After the transformation into polar space, this scale-dependent problem might resolve as demonstrated in the following. The pole of the polar space is the center of the disc (as marked on fig. 3.21a). In the polar space, the circles (so the contour) become rectangular blocks with the height of the whole θ-axis ranging from 0° to 360°[9]. Now, the costs for the graph cut algorithm are $216 = 360 \cdot (1 - 0.4)$ and $108 = 360 \cdot (1 - 0.7)$ respectively. Obviously, the contour of the outer circle has lesser costs now, hence being the optimal contour around the fixation point as expected.

3.7.2 Energy Function

The energy function is the decisive factor for the quality of the graph cut algorithm. It is either based on the Markov Random Field or the Conditional Random Field[10]. The energy function consists of two terms: On the one hand, there is the regional term which describes the likelihood of a node belonging to a certain label and on the other hand, there is the boundary term which describes the coherence between neighboring nodes (see Eq. 3.1). In the graph cut approach, the different terms are incorporated in the weights of the links: The t-links are initialized by using the regional term, whereas the n-links get their weight from calculating the boundary term.

[9] Actually, the range only goes up to 359°, since the angle 0°and 360°are the same.For clarity, this fact is disregarded here.

[10] The Conditional Random Field is actually just a variant of the Markov Random Field where the prior is also based on an observation.

Let every pixel $p \in P = (r, \theta)$ of the edge image I_E^{pol} be a node of the graph G connected to neighboring nodes by a link $(p, q) \in N$ where N denotes a set of all the directly neighboring n-links. The goal of the graph cut algorithm is to find a labeling $X(P) \mapsto l = \{0, 1\}$, $l_p = 0$ denoting 'inside' and $l_p = 1$ 'outside'. The labeling corresponds to the minimum energy where the energy function is defined as:

$$E(X) = \underbrace{\sum_{p \in P} U_p(l_p)}_{\text{regional term}} + \underbrace{\lambda \sum_{(p,q) \in N} V_{p,q} \cdot \delta(l_p, l_q)}_{\text{boundary term}} \tag{3.1}$$

The problem arising from setting the weights of the links connected to the terminal nodes is that the likelihood of the labeling is not known for most of the nodes. Therefore, the weights have to be set to 0 (see Eq. 3.2). The only nodes, which can be definitely labeled, are the ones in the first and the last column. The first column represents the fixation point in Cartesian space and clearly belongs to the object (see Eq. 3.3). The last column contains nodes of which most do not even have valid coordinates in the Cartesian representation, as they are located outside of the image. Therefore, these nodes can clearly be defined as background nodes (see Eq. 3.4). For those nodes, the weights are set to a high value κ to make sure the initial labels do not change as a result of minimization. This approach proposes a value of $\kappa = \lambda \cdot 40$ which results in a sufficiently high value.

$$U_p(l_p) = 0 \qquad \text{if } 0 < r < r_{max} \tag{3.2}$$

$$U_p(l_p) = \begin{cases} 0, & \text{if } l_p = 0 \\ \kappa, & \text{if } l_p = 1 \end{cases} \qquad \text{if } r = 0 \tag{3.3}$$

$$U_p(l_p) = \begin{cases} \kappa, & \text{if } l_p = 0 \\ 0, & \text{if } l_p = 1 \end{cases} \qquad \text{if } r = r_{max} \tag{3.4}$$

The boundary term $B_{(p,q)}$ defines the penalty for neighboring nodes not having the same label. The function $V_{p,q}$ can be almost any cost function and the δ-function removes the whole term if the pixels p and q have different labels a priori. In [Mishra et al., 2009] the cost of assigning a label l_p to a pixel p is based

on the average intensity of the neighboring pixels in the edge map. This present framework uses a different cost function based on the Conditional Random Field as proposed in [Boykov and Jolly, 2001]:

$$B_{p,q} = \lambda \sum_{(p,q)\in N} V_{p,q} \cdot \delta\left(l_p, l_q\right) \tag{3.5}$$

with

$$V_{p,q} = \exp\left(-\beta \cdot I_{E,pq}^{pol\ 2}\right) \cdot dist\left(p,q\right)^{-1} \tag{3.6}$$

and

$$I_{E,pq}^{pol} = \left(I_E^{pol}\left(r_p, \theta_p\right) + I_E^{pol}\left(r_q, \theta_q\right)\right)/2 \tag{3.7}$$

The value for $I_{E,pq}^{pol}$ is calculated the same way as in [Mishra et al., 2009]. The authors also propose different constant values for the other parameters which are exchanged in favor to the ones used in [Peng and Veksler, 2008]. So, $\beta = z \cdot 1/2\sigma^2$ where σ is the global mean intensity of the edge map. The scaling factor $z = 10$ is introduced in this implementation due to the blurred edges, as $I_{E,pq}^{pol}$ returns multiple high values on the edge. $\gamma = 1$ denotes the default value, if pixels p and q have no intensity. $\lambda = 80$ is the weighting factor for the boundary term and is set to a relatively high value according to [Boykov and Jolly, 2001] such that it is more likely to produce an over-segmentation. Since the following grab cut algorithm uses the resulting binary mask to learn the color models, wrongly segmented background regions could grow further in each iteration step. Therefore, it is useful to increase the weight of the boundary term in order to minimize the risk of an under-segmentation. Since an 8-neighborhood is used, the *dist* function includes the weighting for diagonal neighbors in the form of dividing the boundary term by the Euclidean distance between both neighbors. Diagonal neighbors thereby have less influence on the boundary term.

3.8 Grab Cut

The grab cut module provides the filter for refining the rough object mask obtained by the graph cut algorithm. The grab cut algorithm is 'only' an extension of the grab cut approach, thus the graph setup is almost the same. Due to performance reasons, the OpenCV-based method is used. The implementation of the algorithm differs slightly from the standard method: Instead of solely using a trimap for the labeling (object, background and unknown), the 'unknown' label is split into a 'probably foreground' and 'probably background' label. This has several advantages, especially for this framework, which will be discussed later on is this section.

The algorithm itself can be subdivided into multiple steps: The first step is to initialize the color models with the given input mask. All foreground pixel colors are used to create the Gaussian Mixture Model. The first time, a k-means algorithm is used in order to find the different clusters. This implementation uses five components for the model. The background model is created analogously by using the background pixel colors.

The second step is to precompute the weights of the n-links in order to be used in each iteration step. During the iteration, the Gaussian Mixture Models are learned anew by using the given mask. These models are then used in the energy function to construct the graph. Finally, the estimation of the labels is realized by using the min-cut/max-flow algorithm. The mask is updated and the process repeats itself, until convergence or a certain number of iterations has been reached. The goal of the grab cut algorithm is to optimize the labeling results of the graph cut. The original grab cut employs user input to determine background regions, more precisely, the pixels outside of the user-drawn rectangle as shown in Fig. 1.7a of section 1.1.

Due to their high value in the regional term, the background pixels are not likely to change their labels during the iterations. The same counts for preset foreground pixels: The grab cut algorithm marks the inner part of the rectangle as 'unknown'

which is used in order to learn the foreground color model. However, this three-label approach is inadequate for this framework, since it is not possible to make distinct decisions about which pixel belongs to the background or foreground. Parts of the rough object region could still be background pixels and pixels labeled as background could still belong to the object.

The solution to this challenge is to add a fourth label as said before. The mask of the graph cut is now used to initialize the input mask for the grab cut algorithm, where background and foreground pixels correspond to 'probable background' and 'probable foreground' labels respectively. This implies that not any of the pixels is labeled definitely 'foreground' or 'background' such that they are allowed to change their label during the iterations, as the labeling is now constrained by the color models, too.

The use of splitting the 'unknown' label is that 'probable background' or 'probable foreground' pixels can be used to build the two Gaussian Mixture Models for background and foreground. In addition, it is possible to make the same assumption as in the graph cut algorithm: The first row must be located inside the object and gets the label 'foreground', whereas the last row is assigned with the 'background' label. This ensures that the object itself will not vanish due to the iterative optimization.

As a result of labeling most of the pixels 'unknown', the probability of pixels with similar color to the object being classified as foreground pixels rises, even though they are not connected to the object's region. Selecting the 'right' region has to be done separately as described in 3.9.

The critical point of this approach is the size of the rough object region resulting from the graph cut algorithm: If the approximation of the object region is too 'sloppy', i.e. the region is either too small or too large in regard to the object, the result of the graph cut algorithm will generate an under- or over-segmentation.

In the first case, too many object pixels are classified as 'probable background'. This results in 'probable foreground' to be labeled as 'probable background' pixels

in the next iteration step, since the background color model for the object pixels is stronger than the foreground color model.

In the second case, the opposite can happen. Falsely marked background pixels are used to build the foreground color model which causes the foreground region to grow far in excess of the actual object boundary.

However, the advantages of using the grab cut algorithm predominate the drawbacks. The result is a much cleaner object region, especially in areas affected by the 'shortcut' problem described later in section 3.8.2. Moreover, it compensates for the quality loss of the edge map due to performance optimization done in the edge detection: As the color models contain multiple color components, it is possible to extract regions with a textured content to a certain degree. Therefore, an extensive texture edge detection is not necessarily needed. In order to incorporate these advantages into the graph cut algorithm, the used energy function has to be redefined to include the color models.

3.8.1 Energy Function

The energy function used for the grab cut is quite similar to the one for the graph cut algorithm. The main difference lies in the integration of the color information into the energy function. This is done in the regional term. The regional term now depends on the color models C_0 and C_1 for foreground and background respectively, and represents the negative log-likelihood of one pixel belonging to one of the models:

$$U_p\left(l_p\right) = -\log f_{C_{l_p}}\left(p\right) \tag{3.8}$$

where

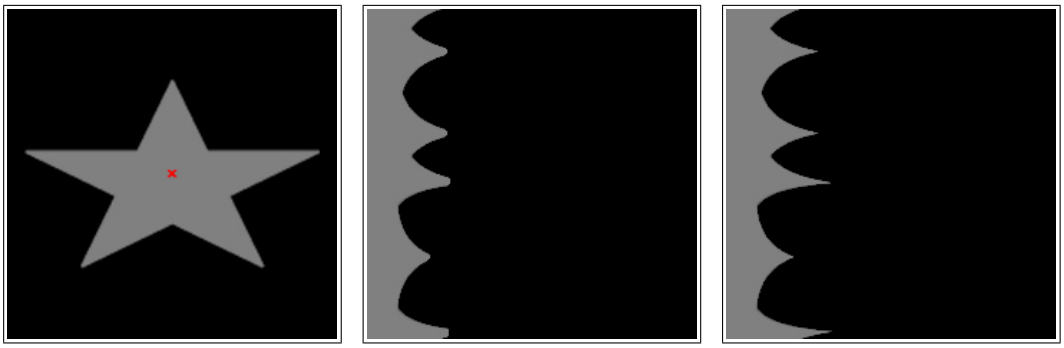
$$f_{C_{l_p}}\left(p\right) = \sum_{i=1}^{K} w_i \cdot g\left(p \mid \mu_i, \Sigma_i\right) \tag{3.9}$$

and $g\left(l_p \mid \mu_i, \Sigma_i\right)$ denoting the Gaussian probability density function (PDF). Hence, the function $f_{C_{l_p}}$ denotes the weighted sum of the Gaussian PDF for each component given a pixel p.

The formulation of the boundary term is equivalent to the Eqn. 3.6 in the graph cut algorithm. The only difference are the parameter settings. As the grab cut algorithm should rely less on the detected edges, but more on the color information, the new value for $\lambda = 40$ and the scale factor z for computing β becomes $z = 0.5$ (cf. section 3.7.2).

3.8.2 The 'Shortcut' Problem

Even though the polar space approach can handle scale invariant contour detections, the used graph cut is still suffering from the so called shrinking bias: Due to the boundary term used in the energy/cost function, the algorithm prefers short boundaries. This can especially be noticeable when the algorithm takes a 'shortcut' through the interior of an object to avoid segmenting an appendage as visualized in 3.22. In Cartesian space the cheapest boundary is actually a circle whereas in polar space, the segmentation is biased towards a straight line along the θ-axis.



(a) **(b)** **(c)**

Figure 3.22: ***(a)*** *A gray star in Cartesian space with the fixation point marked in red.* ***(b)*** *The result of the graph cut algorithm in polar space. Note the cut-off spikes of the star.* ***(c)*** *The result of the grab cut algorithm with an iteration step. The spikes are now included.*

There are two possibilities to counteract this behavior: The first solution would be to loosen the boundary constraints given in the energy/cost function, thus increasing the edge sensitivity. However, thereby the algorithm gets also attracted to weaker edges in general which could lead to errors in other parts of the segmentation. The second and better strategy increases the sensitivity of the color model in the regional term (or decreases the weight of the boundary term respectively) of the cost/energy function. The drawback of this method is that other (background) regions with similar color properties are more likely to be incorrectly segmented as the foreground region. The handling of other unconnected regions would then have to be done separately.

The grab cut algorithm extends the graph cut algorithm exactly in this way by using a Gaussian Mixture Model for each background and foreground color. The algorithm can be used iteratively and for each step, the color models are learned anew and the image graph is built up again with new weights. Thereby, the segmented region can grow steadily into regions with similar colors. Fig. 3.22b shows a star segmented by the graph cut algorithm using this framework. Clearly, the spikes of the star are cut off due to the 'shortcut' problem. When refining this image with the grab cut algorithm (see fig. 3.22c), the region grows into the spikes and makes a correct segmentation.

3.9 Contour Detection

The contour detection module is the final step of the algorithm. It takes the binary mask resulting from the grab cut algorithm, which has been retransformed to Cartesian space, as input and detects all closed contours in that binary image. The algorithm is implemented as described in [Suzuki and Be, 1985]. As the grab cut algorithm can produce more than one contour due to the color models described in section 3.8, it is necessary to find the desired contour. Therefore, this module is also reliant on the fixation point as an input parameter. The point has to be tested against every contour; if it is located inside a contour, it is verified to be the desired object boundary.

In order to reduce the number of contours to test against, an assumption can be made: The object contour has to be a top-level contour, i.e. there is no contour containing the object's contour as a nested one. If a top-level contour does not contain the fixation point, the nested contours will not contain it either. For the case that the fixation point lies within a top-level contour, this assumption is still valid for most cases:

If the point of interest lies within the top-level contour, but not in a nested contour, the top-level contour will be obviously the object's contour and the nested contours will be 'holes' in the object.

If a nested contour, which is a direct child of the top-level contour, contains the fixation point, there will be two possibilities: Either it is the object or it belongs to the background. A contour is defined as the border between background and object regions. Therefore, a nested contour cannot be an object, since it implies that the surrounding top-level contour must enclose a background region. This again is impossible, as the image itself must belong to the object region due to the definition of a contour. The second case, being a background region, is also impossible, as by definition the fixation point lies within the object region.

The only case where a nested contour can be an object is when the contour level is odd numbered where the top-level is defined as the first level, the directly nested

contours as the second, children of nested contours as the third, etc.. The simple solution in this case is using connected components instead of contours. That is if nested contours are detected, the top-level (first-level) contour will be defined as the outer border and the second-level nested contours as the inner borders, namely holes in the object. If there is another contour in the 'hole', it will be treated as a top-level contour again. Therefore, in that special case, both objects will be found.

Chapter 4

Experimental Evaluation

The implemented algorithm is an extension of the approach described by [Mishra et al., 2009] with the intention of using it in a real-time robotic system. In order to optimize the whole segmentation process, the developed framework of this thesis implements a very modular and flexible design and a graphical user interface for a fast visual evaluation as described in section 3.1. This allows to test many different implementations and parameter settings for the segmentation process which can also be changed during the runtime of the application.

However, to evaluate the whole segmentation algorithm, a particular setup has to be chosen. This chapter describes this experimental setup in section 4.1 including the parameter setting as well as the setup of the tested scenes. Afterwards, the algorithm is evaluated in section 4.2 by presenting and discussing the general results. The section 4.3 makes a detailed analysis of the results, presents the advantages and disadvantages of the different involved modules and evaluates the performance of the algorithm in respect to the utilization in a robotic system.

4.1 Experimental Setup

The experimental setup consists of two parts: On the one hand, the algorithm-related parameters have to be defined which includes the computer description, the module arrangement and the precise parameter settings of the different steps of the algorithm. This parameter setup is described in section 4.1.1. On the other hand, the result is also strongly depending on the scene itself. Therefore, the general test environment and different test scenes have to be defined which is covered in section 4.1.2.

4.1.1 Parameter Setup

The parameter setup includes the description of the hardware used in the testing system as well as the general configuration of the algorithm, i.e. the arrangement of the modules. This is described in the first part of this section. Afterwards, the parameter settings of the individual modules are presented.

Hardware Setup

The hardware setup used to evaluate the segmentation algorithm is important when the speed of operation is estimated. Even though the time is not comparable to other implementations of segmentation algorithms, it can be a helpful indicator for the performance of the substeps of the algorithm in relation to each other.
All tests have been done using a notebook computer with an *Intel Core i7-2630QM @ 2GHz* and 4 GB of ram with a *Windows 7* 64-bit operating system. The camera device for video input is a *Microsoft XBOX 360 Kinect-Sensor* with a *RGB* camera and two depth sensors. According to the manual, the optimal operating distance for human body interaction lies in a range from ca. 1.8 to 2.5

meters. According to the developers of the *OpenKinect* library[1], it has a practical ranging limit of ca. 0.8 and 3.5 meters. The output of the camera and the depth sensor are respectively a *RGB* and a monochrome image with a resolution of 640×480 at 30 Hz.

Module Arrangement

The algorithm is implemented as mentioned in section 3.2. The input for the algorithm is the *RGB* image and the disparity map provided by the *Microsoft Kinect* camera. First, the images are resized to a resolution of 480×360 pixels. As the disparity image tends to be very noise, it is 'cleaned up' by applying a closing filter. Then the fixation point has to be selected which is used to transform the images to log-polar space. Afterwards, the *RGB* image is converted to the *LAB* color space. The edge detector uses the disparity map and the *LAB* image to generate an edge map by merging four gradient maps for multiple orientations: the brightness gradient, two color gradients and the gradient from the disparity image. This edge map is then processed by the graph cut algorithm to create a first rough binary label mask. Subsequently, the original polar space-transformed image is converted to *YCC* color space and serves with the binary mask as the input for the grab cut algorithm. It iteratively refines the mask which gets transformed back to Cartesian space afterwards. The contour detection cleans the mask by applying another closing operation and finally returns the contour of the fixated object.

Parameter Settings

The parameter settings of the algorithm are chosen by means of the most promising configuration based on the theoretical preliminary considerations explained in sections 3.3 to 3.9.

[1] See http://openkinect.org for more details.

The image input for the algorithm is provided by the *Microsoft Kinect* camera, as mentioned above. The device returns only low-quality images – both images suffer from strong noise, whereas the disparity map also contains occluded areas due to the infrared sensors. Therefore and due to performance reasons, the image is scaled to a size of 480×360 pixels. Additionally, the disparity map is optimized by a closing filter. The number of iterations and the size of the circular kernel is proportional to the image size: For the given image size, the best trade-off between the stability of the depth boundaries and the quality of details is a 5×5 kernel with five iterations.

For the transformation into circular coordinates, the log-polar transformation is applied to smoothen the textures near the object and providing more color information for the grab cut algorithm at the same time. The cubic interpolation method is used to interpolate the pixels.

The color space for the edge detection is the *YCC* space, as the circular color spaces generate too many image artifact due to the low quality of the *Microsoft Kinect* camera. The same color space is used for the grab cut algorithm.

For the edge detection multiple parameters have to be adjusted. The kernel size is dependent on the image size. In this case, a kernel of size 15×15 is applied for computing the gradient image of all image channels including the disparity map. As the kernel is an oriented filter, the gradient maps of five orientations are merged, as more orientation do not change the result significantly. The color and the lightness gradients are blended with a ratio of 3:2 which is afterwards merged with the depth gradient using a 1:1 ratio.

The settings for the graph cut and grab cut algorithm are all related to the energy function. These settings can be found in section 3.7.2 and section 3.8.1 respectively. The settings for the graph cut algorithm lead to a strong attraction to edges, whereas the grab cut settings induce a stronger weighting of the colors.

The contour detection refines the output mask of the grab cut algorithm before finding the object's boundary. Therefore, it applies a 3×3 closing filter with

only one iteration, as only small holes and frayed edges should be fixed without dislocating the boundary.

4.1.2 Scene Setup

The setup of the scene plays an important role for the image segmentation. On the one hand, there are different constraints due to the hardware, the implemented algorithms and above all the objective for using this framework in a robotic system. These are summarized in the following paragraph of this section. On the other hand, this section describes different scene setups to test the framework in various situations. This allows to evaluate the capability of this segmentation algorithm and to find its limits.

General Settings

The general settings define the constraints of the scene setup. As many segmentation algorithms, this framework is also optimized for the usage in a certain scenario, namely in a robotic system. Therefore, the scene setup has to satisfy a few preconditions: The application is designed to only segment 'simple' objects, as defined in [Mishra and Aloimonos, 2011], because for a robot only objects are relevant which can be interacted with. Moreover, the background of the test scene should kept moderate, as too many different colors will reduce the quality of the edge detector and the grab cut algorithm.

The usage of the *Microsoft Kinect* camera adds additional constraints the scene setup: First of all, the device is designed to work in an indoor environment. The infrared sensor for generating the disparity map cannot handle sunlight[2] very well, as it is dependent on the reflection of the emitted infrared mesh. Therefore, an artificial light source is necessary. The infrared sensor additionally restricts the distance between the camera and the target objects, since the power of the

[2]The natural light partly includes the infrared spectrum.

emitted infrared rays is limited. It has an optimal operating distance limit of ca. 0.8 to 3.5 meters, as stated in section 4.1.1. Therefore, the objects to identify should be arranged within this limit. Besides the restrictions due to the infrared sensor, the *RGB* camera also suggests to be used in a bright environment, as it responds with strong noise generation when applied in poorly illuminated scenes.

Scene Setup

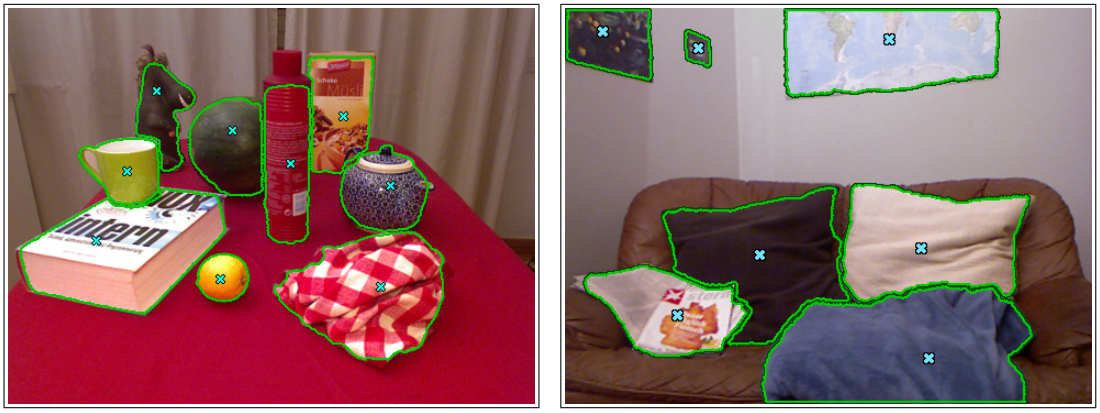
The scene setup has to be described in order to evaluate the functionality of the algorithm and its components. By testing different objects the limits of the proposed segmentation algorithm can be found. This includes

- different simple objects to test and demonstrate the basic functionality.
- small and large objects to test the upper and lower bounds of objects which can be detected. Thereby, the log-polar space module is evaluated, as the capability of segmenting differently scaled object is depending on the scale invariance and the smoothing of the region close to the object.
- object with different shapes like sharp-edged, elongated and hairy objects to test the limits of the fixation-based approach. It shows the effects of the large kernel size of the edge detection on thin object regions. Additionally, the 'shortcut' problem of the graph/grab cut algorithm is evaluated.
- textured objects to test the ability of handling multi-colored objects. Thus, the capabilities of the edge detection with its large kernel size and the grab cut algorithm with its color models are evaluated.

4.2 General Results

The proposed segmentation algorithm is designed to detect 'simple' objects which can be used for robot interaction. Therefore, a scene is set up with a variety of more or less common objects with different properties. The test result of each single object segmentation is summarized in an image composition in fig. 4.1a.

The image demonstrates the capability of the framework to handle various objects. However, most objects in the scene are arranged in a way to generate strong depth information such that the contours are easier to extract. Therefore, another more casual scene is segmented in fig. 4.1b. Instead of using solid objects, this scene shows more soft and flat objects which generate not only varying depth information but additionally have only contact boundaries, i.e. there almost no gap in the depth between different objects.



(a) The image shows various objects arranged on a table.

(b) An every day scene of a couch with pillows and a newspaper lying on it. On the wall there are some pictures and a world map.

Figure 4.1: *The result of different object segmentations composed in a single image.* ***(a)*** *shows the segmentation of mostly solid objects with strong depth information.* ***(b)*** *shows the segmentation of soft and flat objects with weak depth information. The 'x' marks the chosen fixation point for each object respectively. The objects' contours are marked in green.*

In both scenes, all objects are detected very well, even though the contours are partly imprecise, mostly due to the bad image and disparity map quality. The different object properties and segmentations are analyzed in the following.

4.2.1 Simple Objects

The segmentation of simple objects is the basis for testing the capability of the algorithm. If it already fails at this stage, it will probably also fail for more complex objects. Simple object, in this case, are defined as sold objects of constant color. Their shape is convex without having sharp edges. Objects of this category are, e.g., a lemon, an apple, a melon or a ball. A segmentation of simple objects can be seen in fig. 4.2.

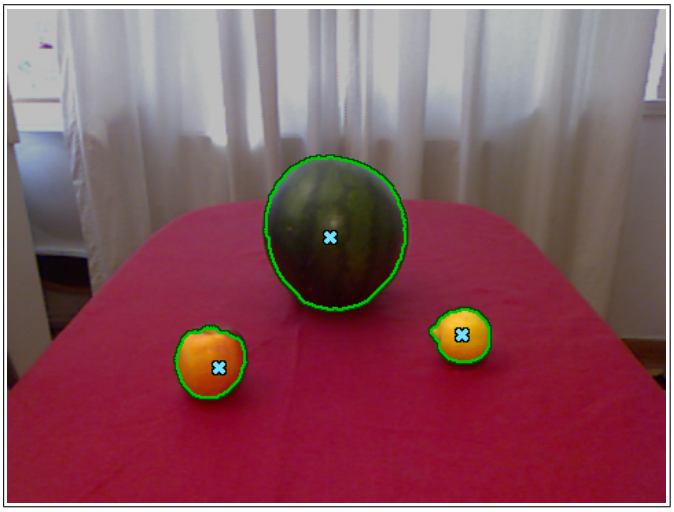
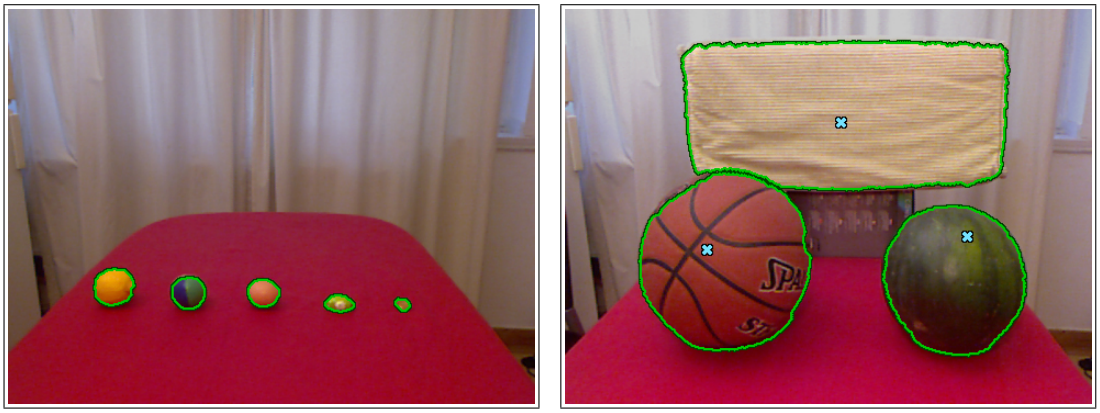


Figure 4.2: *A segmentation of simple objects.*

The three simple objects of different size are all segmented very well. The location of the contours is optimal, except for the apple, as the upper right part is labeled as background. This has two reasons: On the one hand, the red tint of this part is similar to the color of the table cloth. On the other hand, the quality of the disparity map gets worse for relatively small objects as described in detail in section 4.3.1, as there is less depth information available. In combination, this leads to dislocated contours, as the distinction between foreground and background, due to the similar colors, relies mainly on the coarse edge map. This effect can be seen in all segmentation where the colors are similar and the edge map has generated inaccurate edges.

4.2.2 Small and Large Objects

This section describes the impact of the object's size on the segmentation process. Therefore two different scenes are evaluated: One the one hand, the scene contains small simple objects decreasing in size. On the other hand, large simple objects are evaluated in another setup. Simple objects, which are easy to segment, are used, in order to evaluate the size independently from the texturing of the object. Fig. 4.3a and fig. 4.3b show the results of the segmentation respectively.



(a) Small objects: a lemon, a juggling ball, an egg, a chocolate candy and a 10-cent coin. Due to the small size of the objects the fixation point are not shown.

(b) Large objects: a water melon, a basketball and a couch pillow.

Figure 4.3: *A segmentation of simple* ***(a)*** *small and* ***(b)*** *large objects.*

The results of the segmentations produce for both, large and small objects, very good results. Especially the segmentation of very small objects outperforms the expectations: The smallest segmented object is a 10-cent coin in a distance of 80cm and its diameter on the image is only 15 pixels wide. As the fixation point is very close to the boundary, the pixels get smoothed by the log-polar transformation. This leads to an unclean edge such that other edges are preferred during the graph cut algorithm. This problem is discussed in section 4.3.3 in detail. In this case, the colors of the objects are too distinct such that the segmentation results in a correct contour. The coin is the smallest object tested which, in case of robot interaction, is sufficient, as even smaller objects are not useful anymore.

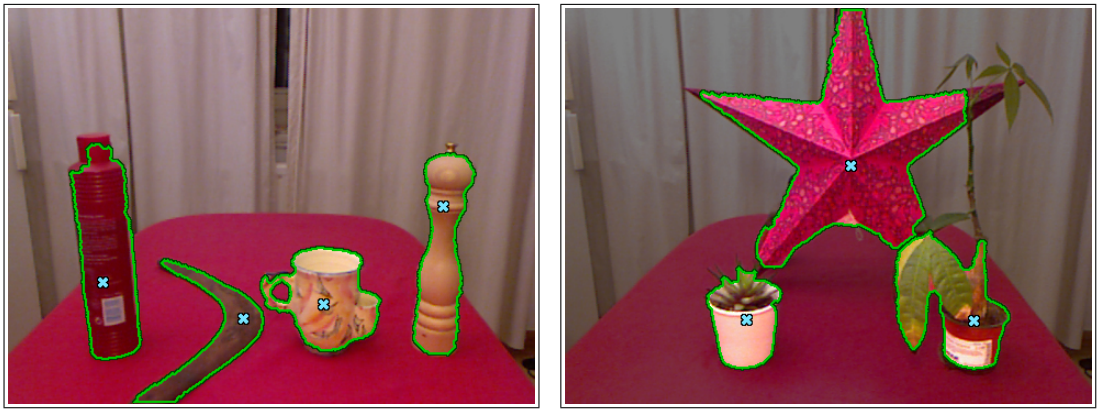
Large objects can be segmented much easier. As the fig. 4.3b shows, the upper limit for the object size is sufficiently large. A segmentation of an image-filling object does not make any sense. The only incorrect segmentation occurs near the boundary of the basketball where the contour is dislocated at some parts. This has one major reason: The edge detection of the disparity map results in an inaccurate border. This is mainly caused by the bad quality of the disparity map which is discussed in section 4.3.1 in detail.

Another issue regarding the basketball's contour is the internal texture. The stripes and the font create strong color edges. These cannot be removed by the large edge detector kernel, as it is not a small texture pattern. The only way to remove such textures is a stronger weighting of the disparity edges. Finding the 'right' mixing function between color and depth edges is generally not possible. Both problems, the influence of the kernel size and the weighting of color and depth information, are discussed in section 3.6.

4.2.3 Objects of Different Shapes

The shape of an object can have a large impact on the segmentation result. In the following, the segmentation results of objects with different shapes are presented. This includes objects with sharp-edges, elongated objects, thin and hairy objects. It addresses in particular the behavior of the graph cut and grab cut algorithm, as they are most sensible to different shapes. Additionally, it tests the large kernel size used for the edge detection and the precision of the disparity map.

Fig. 4.4 shows a segmentation of objects with shapes which are more difficult to segment, especially in regard to the graph cut-based algorithms. When transformed to log-polar space, the shown objects in the scene are deformed to long horizontal objects, as they are bend around the fixation point. However, the graph cut algorithm tries to find a 'short' vertical cut to minimize the energy. Therefore, long objects and objects with sharp edges tend to get cut off. The grab cut algorithm suffers from the same problem, but it can counteract this behavior



(a) Different elongated objects and a uncommonly-shaped mug.

(b) Objects with complex shapes: a plant with a thin caulis, a star-shaped lamp and a small aloe vera.

Figure 4.4: *A segmentation of objects with* ***(a)*** *elongated and* ***(b)*** *more complicated shapes.*

by using the color information as described in section 3.8.2. In fact, the grab cut approach of this framework sets in at exactly this point. It allows to optimize the result of the graph cut algorithm, especially in these cases, immensely as shown in section 4.3.4.

Even though the grab cut algorithm is able to refine the contours of the objects, it still has its limits. If the object is too thin, i.e. the edges of the boundary are too close, this method will not work, as seen in the segmentation of the plant with the thin caulis and the endings of the spikes of the star-lamp in fig. 4.4b. The difficulties arising when segmenting objects with thin or spiked shapes are caused by multiple reasons. First, the resolution of the disparity map is not sufficient to detect very thin objects. This is discussed in section 4.3.1. Second, elongated objects are deformed during the log-polar transformation such that have an unfavorable shape for the graph-based algorithms as described in section 4.3.2. Last, the edge detection blurs edges during the detection process such that closely edges tend to grow together. This effect is explained in section 3.6. A similar effect occurs when the dislocated edges get too strong like in the segmentation of the red bottle in fig. 4.4a and of the aloe vera in fig. 4.4b which is discussed in the same section.

4.2.4 Textured Objects

Textured objects are the most difficult objects to segment, especially using this framework, where the computation of the texture gradient is neglected to increase the performance of the edge detection. As the missing texture gradient is compensated by the large kernel filter in the edge detection and the color models in the grab cut algorithm, it is important to evaluate, how the proposed framework can handle textured objects. Therefore, a scene with different types of textures is set up including periodical textures of different size and irregular textures with text and pictures. Fig. 4.5 shows the segmentation of differently textured objects.

Figure 4.5: *A segmentation of textured objects.*

In general, textured objects are hard to segment for a generic reason: It is difficult to tell whether a texture is a pattern which should be ignored or an own object. That means, when e.g. a magazine has a smaller picture on it, the question is whether the picture is an object itself or just part of the magazine. Given a single fixation point, this question cannot be answered. The proposed solution of this algorithm is to ignore small patterns and to segment larger ones. The threshold is depending on the internal color edges of the object and the selection of the fixation point, as analyzed in section 4.3.4.

Even though the textures are not handled explicitly, the segmentation of the objects in fig. 4.5 show for all types of textures very good results. The only

difficulties occur close to the dish towels. The red and white camouflaged dish towel includes a small table region due to the similar color of the table cloth, whereas in the contour of the green-blue dish towel, where the texture pattern is very large, the bottom part of the towel is missing due to the strong internal edge. These two phenomena show the limits of the different weighting of the color and edge information used in the grab cut algorithm which is analyzed in detail in section 4.3.4.

4.3 Analysis

The presentation of the results showed that the proposed algorithm is capable of segmenting a variety of objects very concisely. In the case of the evaluated objects, segmentation errors are largely restricted to minor boundary defects. They are mainly caused by the bad quality of the retrieved disparity map from the *Microsoft Kinect* device and the consequential parameter settings of the algorithm. Especially the weighting of color, lightness and depth edges in the edge detector and the weighting of the regional and boundary term in the grab cut algorithm are the critical parameters of the algorithm. Despite this, the results show that the approach of a more balanced algorithm – that is in which the grab cut is improved – works as expected. The grab cut is capable of handling most of the errors occurring during the preceding steps.

This section analyses the results in detail: The occurred errors are explained in regard to the intermediate steps of the algorithm and the advantages of the different modules are evaluated in sections 4.3.1 to 4.3.4. Besides evaluating the effectiveness of the algorithm, the efficiency of the algorithm is covered in section 4.3.5. There, the computational complexity is analyzed by measuring the runtime and comparing it to the original approach of [Mishra et al., 2009].

4.3.1 Disparity Map

One critical point in this approach is the use of the disparity map which is provided by the *Microsoft Kinect* camera. As stated in section 3.4.3, the quality of the disparity map is very low, as it suffers from noise, occlusions and reflections. This leads to dislocated depth information, especially close to the boundary of objects.

Therefore, this framework proposed a 'cleaning' of the disparity map by using the morphological closing filter. The result is a stabilized disparity map with almost no undefined areas, i.e. the regions where no depth information is available is

reduced. In the original map, these regions are left black. The cost of stabilizing the disparity map is the loss of detail. This means especially that small and thin objects loose their depth information. However, in contrast to the gain, this is a reasonable trade-off. Fig. 4.6 shows different disparity maps and their 'cleaned' versions. For most objects, this operation results in clean depth edges such that the edge detection does not find double edges.

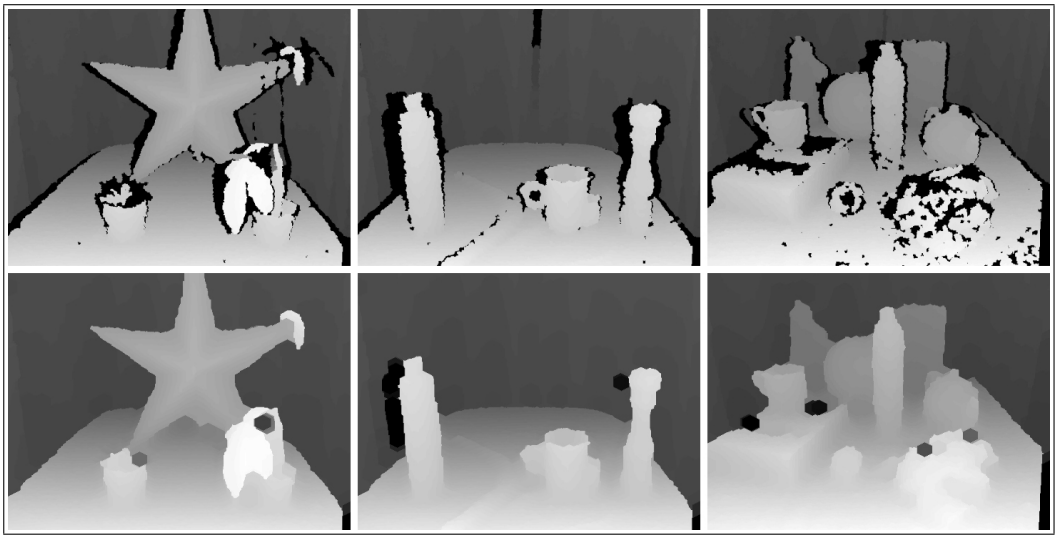
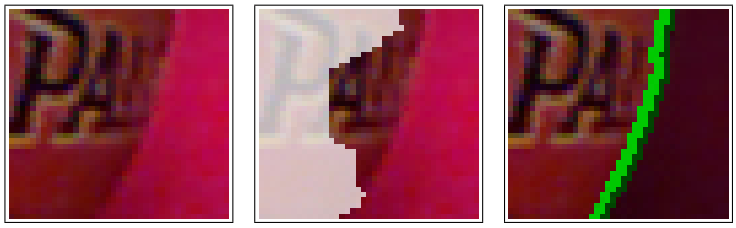


Figure 4.6: *The effect of the 'cleaning' procedure of the disparity map. The top row shows the original disparity maps, whereas the bottom row shows the 'cleaned' results respectively.*

Note that even though the black areas vanish, the brighter depth information stays in place. This implies that also the defects of depth boundaries will not be repaired by this procedure, as seen on the first column in Fig. 4.6, where the leafs of the aloe vera (bottom left) disappear, as there is no depth information available. Due to the rough resolution of the *Kinect*'s depth sensor, the plant's caulis (same image, bottom right) is not even recognized. Only a suggestion of an occlusion, barely visible as the vertical black region, is given which disappears after the 'cleaning' process.

As the errors in the disparity map are passed to the following edge detection, this results in a wrongly detected edge. But the edge of the disparity map is mainly used to enhance the other edges, as it is weighted less, the defects are not propagated to the graph cut algorithm such that it finds the correct contours.

Fig. 4.7 shows an example of a strong error in the disparity map, where the graph cut algorithm finds the correct contour nevertheless.

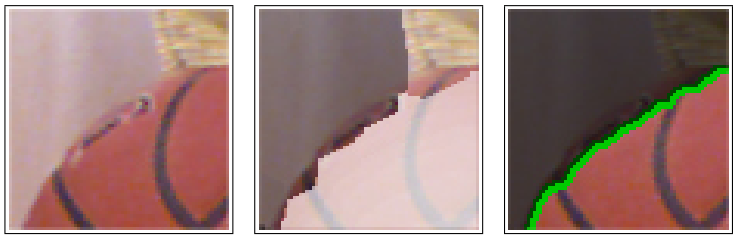


(a) A part of a basketball. **(b)** The accordant 'cleaned' disparity map. **(c)** The computed boundary.

Figure 4.7: *A segmentation with an incorrect disparity map. The error of the disparity map has no effect on the graph cut algorithm.*

However, even though the edge detection, the graph cut and the grab cut algorithm can handle minor errors of the disparity map, there is a special case when the refinement by the grab cut reintroduces the error. This effect can be observed especially in low-quality images with strong noise where the color values near object boundaries vary more.

The preconditions are the following: On the on hand, the disparity map has to generate a dislocated depth edge. On the other hand, the colors of the object and the background close to the boundary are similar. This often occurs, especially at the boundary region, due to low-quality images where object edges get blurred because of noise or diffuse light.



(a) A part of a basketball. **(b)** The accordant 'cleaned' disparity map. **(c)** The computed boundary.

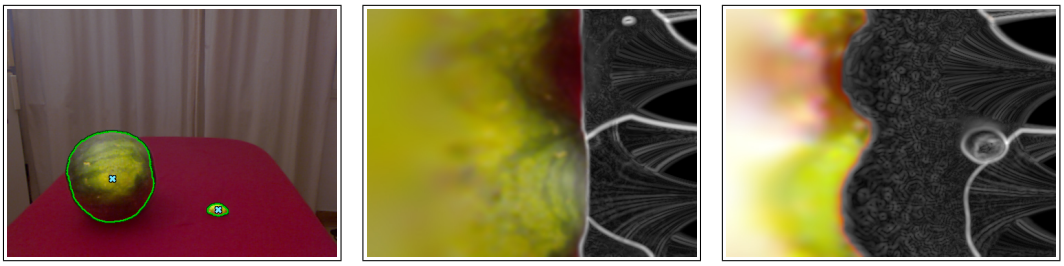
Figure 4.8: *A segmentation with an incorrect disparity map. The error of the disparity map is reintroduced by the grab cut algorithm.*

Now, when observing a pixel near a boundary on the dislocated edge, the membership to the background color model might increase due to the similar colors

and the additional noise. If the dislocated depth edge is strong enough, the energy for the pixel becomes less for being a background pixel. Therefore, the new boundary proceeds along the dislocated depth edge. The fig. 4.8 shows this effect in case of a basketball. Note that the boundary area of the basketball is much brighter than the rest due the diffuse light and the imprinted text generates an additional internal color edge as well. However, as seen in the results in section 4.2, this consequences of this effect are in most cases inexistent or negligible.

4.3.2 Log-Polar Transformation

The log-polar space is the key for a fixation-based segmentation. The advantages have been discussed in section 3.5 and section 3.7.1. The scale invariance property allows to segment objects of different sizes equally. This has been tested in section 4.2.2. Fig. 4.9 shows a comparison of two different sized objects and their edge detection in log-polar space, respectively. Even though the objects have opposite sizes in Cartesian space, they look very similar in log-polar space. The main difference is a little shift along the horizontal axis.



(a) A big melon and a small candy.

(b) The segmentation of the melon in log-polar space.

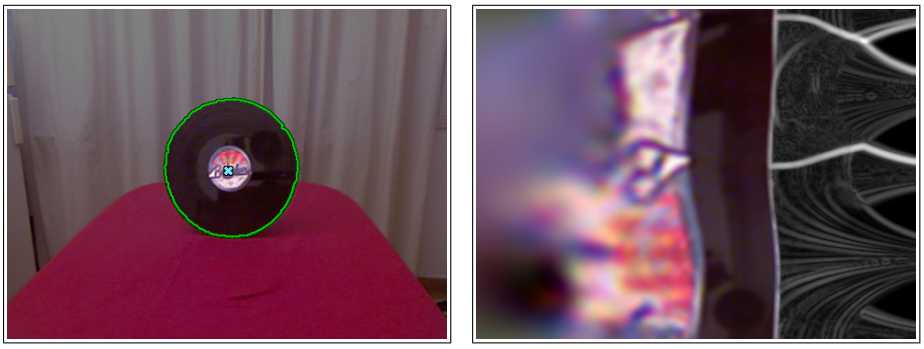
(c) The segmentation of the candy in log-polar space.

Figure 4.9: *The transformation of different sized objects into log-polar space. The similar shapes are both located near the center.*

This property is also used to enhance the detection of true object boundaries. If an object has weak closed internal edges due to textures, the graph cut algorithm ignores this contour when there is another stronger closed edge with a larger radius. The effect can be seen in fig. 4.10 where a disc is segmented. The texture on the disc generates an internal edge, but as the true boundary has higher

values, the disc is segmented correctly. This property of the log-polar space thereby supports the segmentation process immensely.

Even though the log-polar transformation contributes a major part to the segmentation process, the use of the log-polar space also has some disadvantages. As shown in fig. 4.9c, the original shape of an object gets deformed in log-polar space.

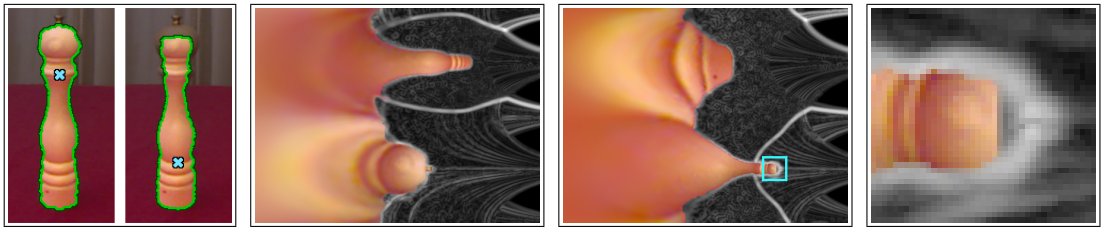


(a) A scene with a segmented disc.

(b) The disc in log-polar space showing the internal and the true boundary of the object.

Figure 4.10: *A segmentation of a disc with an internal edge. Due to the scale invariance the disc is segmented correctly.*

This leads to two major effects: On the one hand, sharp-edged and elongated objects form longer horizontal regions in the log-polar space, where the sharp edges additionally get thinner as they are compressed along the vertical axis. The following graph cut algorithm is not capable of handling such objects, as it is set up to prefer a 'short' vertical cut. This leads to cut off regions when the distance between two edges of an object get too close. As the log-polar transform depends on the fixation point, as it is the pole of the coordinate system, the selection of this point can increase this problem additionally. Fig. 4.11 shows this effect on the basis of an elongated object.



(a) A segmented pepper caster with different fixation points.

(b) The segmentation of the left image (a) in log-polar space.

(c) The segmentation of the right image (a) in log-polar space.

(d) A closeup of the marked region in (c).

Figure 4.11: *The effect of different fixation point selections on an elongated object in log-polar space. Depending on the fixation point, parts of the object are cut off.*

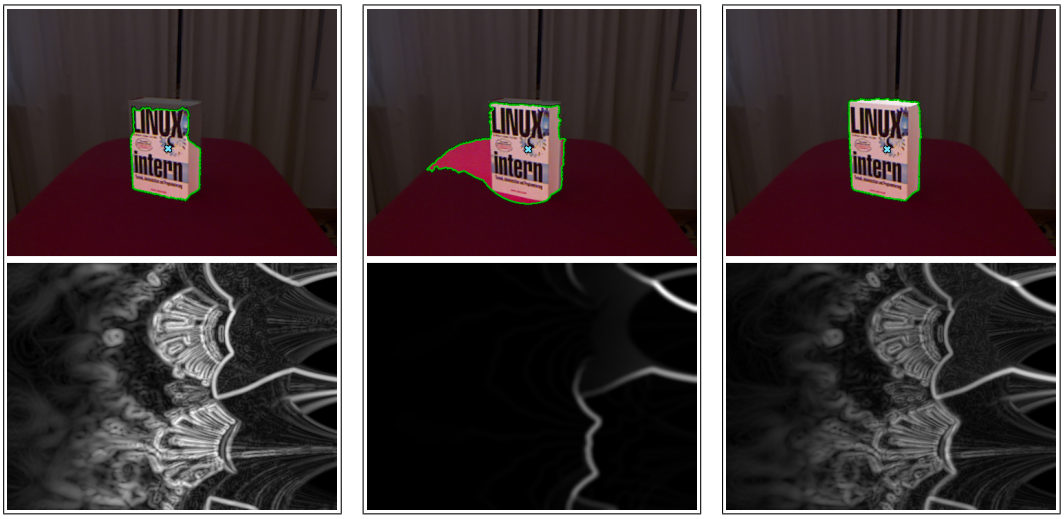
The selection of the fixation point at the bottom of the pepper caster leads to an incomplete segmentation of the object. The log-polar transformation shows that the top of the caster is stretched along the horizontal and compressed along the vertical axis. As the edges get closer, even the grab cut algorithm takes the 'shortcut' as seen in fig. 4.11d. This problem leads back to the general problem of the weighting between the regional and the boundary term of the energy function. It is not possible to find universal parameter settings, as a change would only introduce errors in other cases (cf. section 4.3.4).

4.3.3 Edge Detection

The edge detection is the critical step in the approach of [Mishra et al., 2009] and therefore consumes most of the computational runtime. Even though this framework presents a more balanced algorithm, the edge detection is still plays an important role in the segmentation process, as it is the basis for both the graph cut and grab cut algorithm. Compared to the approach of [Mishra et al., 2009], this framework simplifies the edge detection to a minimum, as explained in section 3.6.

An important setting is the mixture of the color/lightness and the depth information. On the one hand, the gradients computed from color and lightness information result in an accurate edge map. On the other hand, these maps also contain a lot of edges due to the texturing of the object and general noise.

The gradient calculated from the disparity can be used to suppress these internal edges, as they denote not the true boundary of the object. However, the disparity map often is very coarse such that the detected edges are dislocated which can lead to a wrong segmentation, as described in section 4.3.1. Additionally, the contact boundaries, the boundary between the object and the surface it is standing on, are not detected. Therefore, the detection of these edges relies on the color information only. This implies that setting up a global optimal parameter is not possible, as there are always scenarios where a stronger color edge is needed or a stronger depth edge. Fig. 4.12 shows edge maps with different weightings to illustrate this problem and explains the errors in the segmentation.



(a) An edge map based on color only.

(b) An edge map based on depth only.

(c) A mixed edge map with a color-depth ratio of 3:2.

Figure 4.12: *The effect of different mixing weights for the color and disparity edge map.*

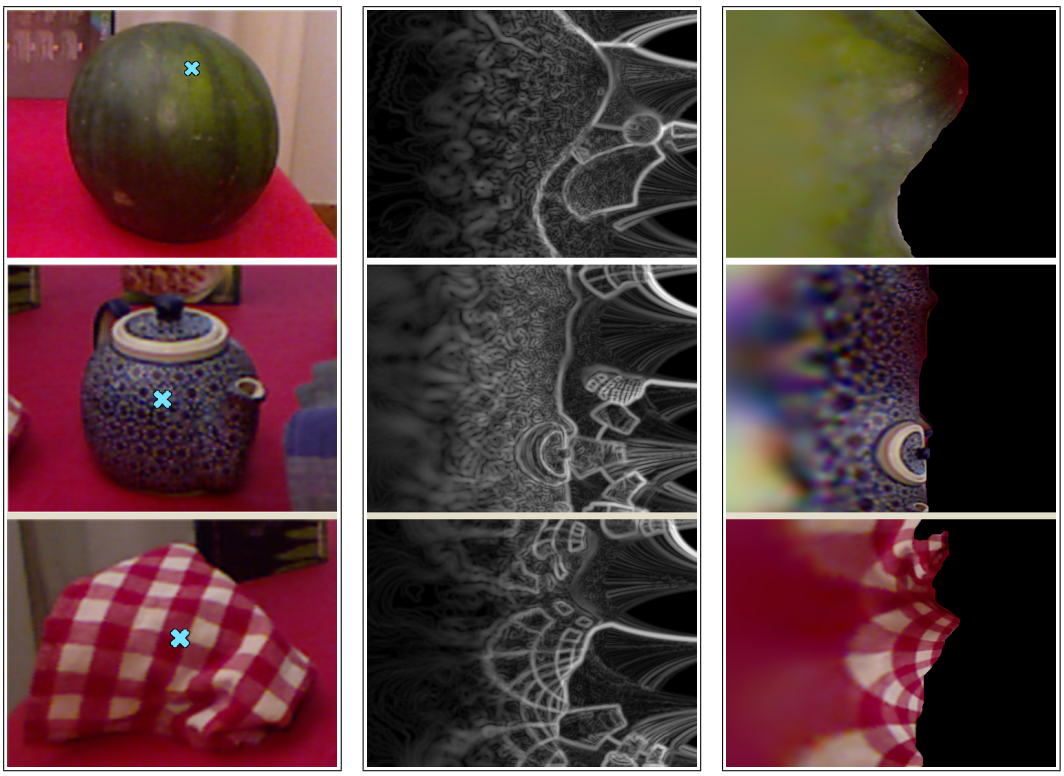
Note that the number of iterations of the grab cut algorithm is reduced to one, as the grab cut would have fixed the wrong segmentations. Fig. 4.12c shows that the internal edges are still strong due to the lower weight of the disparity edge map. The two reasons for choosing a low weight for the disparity edge map are mentioned above: On the one hand, the disparity map is too coarse such that the wrong depth edge would be traced instead of the true color edge. On the other hand, the contact boundary is not detected such that the object area floods into background areas (see fig. 4.12b), especially when the color are similar.

Another feature of the edge detector used in this framework is the large kernel size for smoothing noise and textures with small patterns. The smoothing kernel works in combination with the log-polar transformation and the disparity edge which all together create a powerful tool to reduce the effect of texture patterns and noise without the implementation of an explicit texture gradient map.

The log-polar transformation interpolates the pixels close to the fixation point, thereby smoothing the image inversely proportional to the distance from the pole. The edge detection, which is carried out afterwards, smooths the log-polar transformed image again such that noise or small texture pattern get almost evened out close to the pole. The intensity of the textures is additionally decreased in combination with the disparity edge map. This is the best precondition for the graph cut algorithm which does not need to label the whole noisy or textured area, but an area large enough to represent all different colors of the pattern or noise. As long as the grab cut algorithm can build adequate color models to represent these patterns and as long as there is a strong true boundary enclosing the object, the grab cut algorithm will optimally segment the object. Fig. 4.13 shows the result of the cooperative process. The analysis of the grab cut algorithm is separately covered in section 4.3.4.

Note that the edge map of the melon and the teapot generate similar internal edges, as the color images of the *Microsoft Kinect* contain many color artifacts caused by noise. Even though all textures generate many internal edges, the graph cut algorithm performs very well and covers a sufficient large region to model the colors for the grab cut algorithm.

The large kernel size of the edge detector therefore contributes to the segmentation of textured objects, also on images exposed to strong noise. However, it also leads to several minor disadvantages. As stated before, the large kernel and the log-polar transformation blur the image most close to the pole of the log-polar space. This implies that the selection of the fixation point has a major impact on the grab cut algorithm, as the most accurate color models are generated with samples close this point. Therefore, the location of the point can influence the

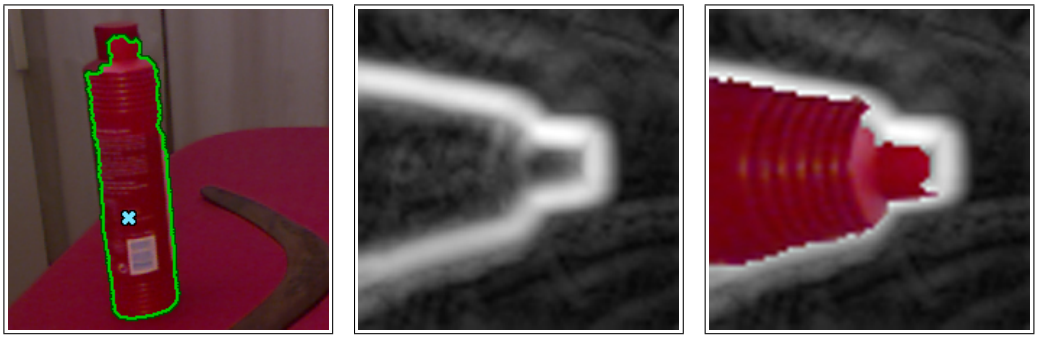


(a) Three objects with different textures. **(b)** The edge maps. **(c)** The graph cut results.

Figure 4.13: *The effect of the combined blurring of the edge detector kernel and the log-polar transformation on the graph cut result. The edge maps show from top to bottom: Internal edges due to noise, a small-scaled texture pattern and a large-scaled texture pattern.*

outcome of the segmentation drastically which is analyzed in section 4.3.4. Additionally, due to the strongly blurred edges, the selection of a fixation point near an object boundary will lead to a wrong segmentation as well, as the boundary edge is less likely to be detected by the graph cut algorithm.

Another problem occurs in cases where strong edges are lying close together. As those edges are smoothed, the area between those edges get an high 'edge value' as well. This increases the 'shortcut' problem in log-polar space described in section 4.3.2. Fig. 4.14 shows an example of this effect occurred during the evaluation.



(a) A segmented object with a defect in the contour. **(b)** The edge map of the defective region. **(c)** The edge map and the grab cut result.

Figure 4.14: *The effect of the strong blurring of edges at close quarters which leads to defects in the contour.*

As seen in Fig. 4.14c, the blurring of the strong edges leads to a cut-off region of the bottle. However, in most cases, the grab cut algorithm is able to optimize the result by the incorporation of the color information.

4.3.4 Graph Cut and Grab Cut

The graph cut algorithm is an intermediate step for the grab cut algorithm. It prepares a rough mask which is used as the input for the following grab cut algorithm. The parameter of the energy function are designed to produce an over-segmentation, whereas the grab cut parameters give more weight to the color information and thereby floods into thin regions ignoring weak edges. As these algorithm are supposed to work as an unit, they have to be analyzed together.

The graph cut algorithm uses only the generated edge map for detecting the object region. This region has to cover an area large enough to build solid color models in the grab cut algorithm. This process is supported by the edge detection as described in section 4.3.3. When the region is too small or too large, the grab cut algorithm can fail in detecting the correct contour. However, the grab cut algorithm in this framework is very tolerant to minor and partly also to major errors occurring in the graph cut algorithm.

The parameters of the grab cut algorithm are especially designed for refining the results of the graph cut algorithm. This approach works on small scale for fine-tuning the contours as well as on large scale where the foreground is cleaned from whole background objects. The effectiveness of this new approach is shown in fig. 4.15.

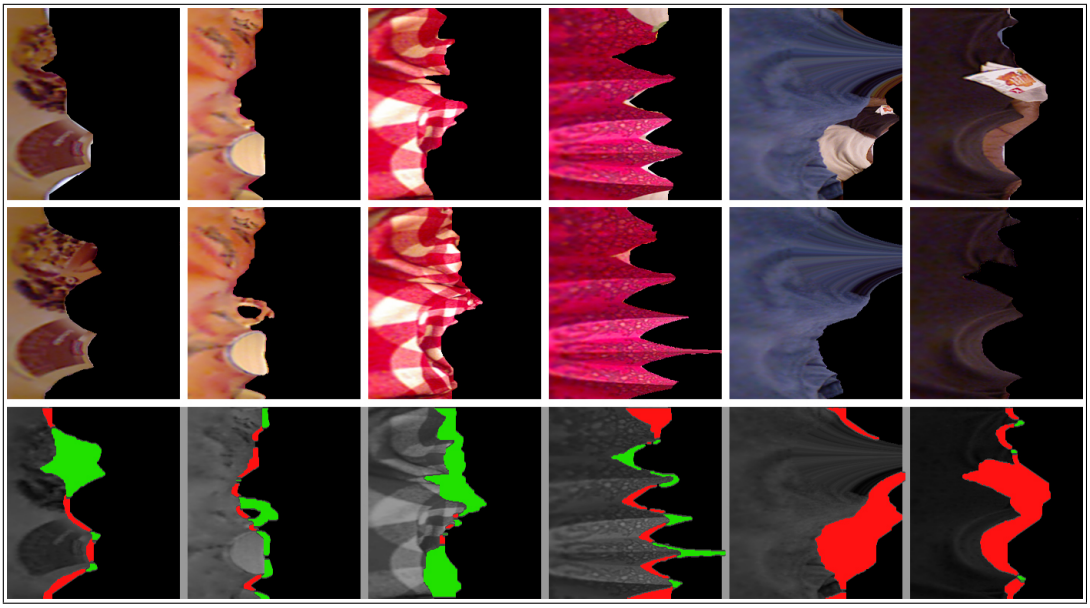
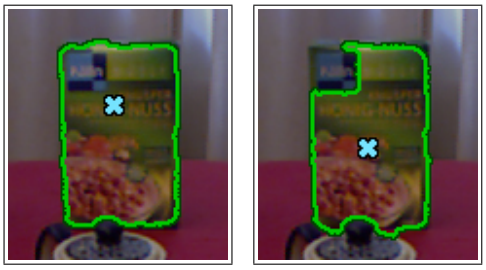


Figure 4.15: *The refinement of the graph cut results after applying the grab cut algorithm with five iterations. The top row shows the graph cut results. The middle row shows the grab cut results. The bottom row shows the added regions in green and the subtracted regions in red.*

The major challenge of this grab cut-based approach is similar to the color and depth edge weighting problem described in section 4.3.3: The selection of the parameters for weighting the regional and the boundary term of the energy function is not trivial. When the regional term is weighted more, the algorithm tends to cluster regions with similar colors (cf. fig. 4.13b). This can lead to an underseg-

mentation, as weak boundary edges are ignored, especially when the object and the background have similar colors. In the opposite case, when the boundary term has a stronger weighting, the edges have a greater influence on the energy, which leads to the 'shortcut' problem (cf. fig. 4.13a) as discussed in section 3.8.2.

Another aspect of applying the grab cut algorithm in log-polar space is the dependency on the selection of the fixation point, as mentioned in section 4.3.2. The color models of the grab cut algorithm are initialized with the sample pixels of the graph cut result. As the log-polar space stretches and interpolates many pixel values close to the pole, the most accurate color models are built on these pixel information, since they all have similar values. This implies that colors more distant to the pole, even if they are in the initial foreground region, are not represented well in the Gaussian Mixture Model. Hence, the grab cut algorithm becomes context-sensitive, i.e. the area around the fixation point defines the importance of the colors. This can lead to undesired effects when segmenting objects with large-scaled textures, i.e. textures where colors do not occur in small periodical intervals, but isolated in different areas of the texture. A correct and a 'mislabeled' segmentation is shown in fig. 4.16.

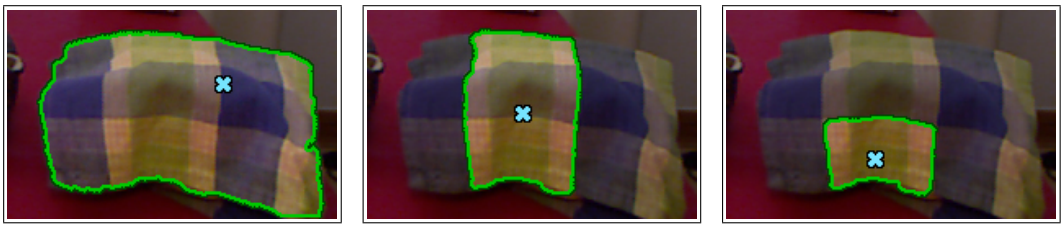


(a) A correct segmentation. **(b)** A 'mislabeled' segmentation.

Figure 4.16: *The effects of the selection of different fixation points.* ***(b)*** *shows an undesired excluded region due to the fixation point selection.*

Even though the image in fig. 4.16b shows an undesired segmentation, the question whether the segmentation is correct or incorrect cannot be answered in general. In fact, the human eye is also not able to anticipate the segmentation of an object with a large-scale texture given only one fixation point. E.g., if someone points at a magazine with a picture on it, it is not clear whether the magazine

or the picture itself is the point of interest. The same accounts for two different objects like a poster on a wall. In order to segment an object or texture correctly, some context is needed. In this approach, a spacial context is given by the fixation point as mentioned before. Fig. 4.17 shows this behavior by the means of a dish towel.



(a) Segmentation of a dish towel. **(b)** Segmentation of a green stripe on the dish towel. **(c)** Segmentation of a green square on the dish towel.

Figure 4.17: *The effects of the selection of different fixation points. The object segmentation depends on the context of the fixation points.*

The size of the context is given implicitly by the log-polar transformation, the edge detector kernel size and the intensity of the internal edges. This behavior of the grab cut algorithm in log-polar space offers an interesting basis for future work.

4.3.5 Computational Complexity

The computational complexity of the presented algorithm is an important factor for its use in a robot system. One of the main objectives is to implement a system which can be run in real-time: The definition of real-time in this context must not be confused with the real-time definition often used in image processing where an image is processed within milliseconds. In this context, real-time is defined as the time it takes a person to identify an unknown object. This duration varies depending on the object and the environment and can take up to some seconds.

Moreover, the measured time for the segmentation process depends much on the hardware used. Therefore, the evaluation presents the absolute duration of the algorithm on the system described in section 4.1.1. Besides that, it also compares the relative durations of the different steps of the algorithm which are

then compared to the relative durations of the original approach by [Mishra et al., 2009].

The measurements are taken for the whole algorithm as well as for each step: The edge detection, the graph cut and the grab cut algorithm. Tbl. 4.1 shows the average values of ca. 100 segmentations on a medium sized object[3].

Module	Duration [sec]	Duration [%]
Overall time	6.27	100.00
Edge Detection	0.40	6.38
Graph Cut	2.27	36.20
Grab Cut	3.46	55.18
Cleaning	0.14	2.24

Table 4.1: *The measured absolute and relative duration for the different steps of the proposed algorithm.*

The whole algorithm takes only about six seconds to compute the segmentation of an object. This is already a good basis for the use in a robotic system. The result of the optimization of the edge detector has to be pointed out in particular. In this approach, it takes less then half a second to compute the lightness, color and disparity edge map. The most time consuming steps are the graph cut and the grab cut algorithm. The graph cut has the most work to do, as the minimum cut calculation is based on mostly unknown labels. In total, the grab cut takes longer than the graph cut. However, it has to be taken into account that the grab cut's total time includes four iterations, including the learning process of the Gaussian Mixture Models. This means that a single iteration of the grab cut algorithm takes less than a second. It performs much better than the graph cut as the input mask is already an approximation of the final result.

The algorithm mainly is a single thread application (except for the image grabbing routine) and uses only 21% of the CPU capacity. The memory consumption varies around an average of 100 MB, when, besides the video player and the output

[3]The runtime of the min-cut/max-flow algorithm used in the graph cut and grab cut algorithm depends on the weights between the nodes. Large areas with similar intensity values take longer to compute as smaller regions with strong intensity changes.

window, no additional observation window is opened. The low CPU consumption is a good starting point for further optimizations.

As the proposed algorithm is an optimization and extension of the algorithm described by [Mishra et al., 2009], a comparison between the two of them is evident. A direct comparison is difficult, as the computational complexity of the original algorithm is not given in detail in the author's paper or related papers. Nevertheless, by taking the graph cut algorithm as a reference point, a comparison can be made to roughly classify this framework. Therefore, the given duration of the original[4] algorithm are shown in tbl. 4.2.

Module	Duration [sec]	Duration [%]
Overall time	∼40.00	100.00
Edge Detection	∼6.00	15.00
Graph Cut (x2)	∼2.00	5.00
Optical flow	∼24.00	60.00
Others	∼8.00	20.00

Table 4.2: *The indicated duration for the different steps of the algorithm proposed by [Mishra and Aloimonos, 2011].*

The graph cut algorithm is computed twice in the algorithm. The second time, a 3D-histogram is used to incorporate the color information into the graph cut algorithm. Therefore, it seems that the graph cut implementation takes about one second to compute on the used system, as described in section 4.1.1. By using the graph cut duration as a base unit, it is possible to compare at least the algorithm of the edge detection[5]:

The comparison of the durations in tbl. 4.3 arrives at the conclusion that the implementation of the present framework runs more than seven times faster than the algorithm proposed by [Mishra and Aloimonos, 2011]. Even though this calculation depends on many unknown variables, at least the edge detection can

[4] A direct comparison is not possible, since their publication does not evaluate the runtime. Therefore, a similar implementation of the same authors is used [Mishra and Aloimonos, 2011] which is also based on the same approach.

[5] 'Gro12' refers to the implementation of this framework, whereas 'MA11' refers to the implementation of the approach by [Mishra and Aloimonos, 2011]

Module	Gro12	MA11	Ratio
Overall time	2.76	20.00	7.25
Edge Detection	0.18	3.00	16.67
Graph Cut	1.00	1.00	1.00
Other	1.58	16.00	10.13

Table 4.3: *The duration of the implementations as multiples of the graph cut duration and as a factor of the performance gain for the proposed implementation.*

be roughly compared, as the implementation is the same as in [Mishra et al., 2009]. The performance gain for the edge detection is even higher: The edge detector is more than 16 times faster than the original approach. Using a faster desktop computer, the proposed algorithm can be run in less than five seconds, which is nearly the time a human being needs to 'segment' an unknown object in an unknown scene.

Chapter 5

Conclusion and Future Work

In this thesis, a new segmentation framework has been proposed using a fixation-based approach first introduced by [Mishra et al., 2009]. In contrast to classical segmentation methods where an image of a scene is divided into multiple individual regions, the fixation-based approach redefines the segmentation process by imitating the human visual system, thus it only separates the fixated region from the rest of the image. This formulation of the segmentation problem is, unlike the classical approach, a well-posed problem, as it regards the interest of the observer in the form of a fixation point.

As this new approach was aiming to be used in a robotic system, the question rose: How can this approach be applied on a real-time robotic system? The critical point of this question is the performance of the segmentation algorithm. As the original approach of [Mishra et al., 2009] does not offer a solution which can be applied in real-time, a way of optimizing their approach had to be found. By examining their work, the edge detection, which is used for the contour detection, was identified as the computational most complex step of the algorithm, as their segmentation results rely highly on a precise edge map.

For optimizing this step, the presented framework in this thesis expands the concept of [Mishra et al., 2009] by introducing the grab cut algorithm as an additional refinement step. In order to incorporate the grab cut algorithm, this

thesis examined how to bridge the information gap between the given fixation point and the information needed for the grab cut algorithm to produce correct segmentation results. The solution was to include the original method as an intermediate step of the new algorithm.

The reformulation of the original approach had extensive consequences for the different steps of the algorithm: The grab cut algorithm uses sophisticated color models for the contour detection process and thereby is very robust to an imprecise input. This allowed to reduce the strong dependency of the original algorithm on the quality of the computed edge map, thereby creating a more balanced algorithm. Consequently, the new formulation opened up the possibilities of optimization.

This potential was tapped for the design of the different steps of the algorithm. In particular, the edge detection was heavily optimized by simplifying the calculation of the color, lightness, texture and depth gradient maps. Moreover, the edge detection was moved into polar space, where the polar space transformation was exchanged in favor to the log-polar transformation.

This combination of changes supports the overall process and leads to an even more balanced algorithm: By exchanging the polar transformation with the log-polar transformation, the pixels close to the fixation point get heavily blurred due to the interpolation occurring during the transformation. As the edge detection is now carried out afterwards, it is less likely to find strong edges close to the pole which are, in fact, mostly internal edges. This in turn boosts the following graph cut algorithm, as it generates a larger object region due to the log-polar space on the one hand and due to the weaker internal edges on the other hand. A larger object region, again, is preferable for the grab cut algorithm, as it can build a more precise color model for the object in consequence of more available pixel values.

Another advantage of edge detection carried out in log-polar space is that it is equal to the use of an edge detection kernel in Cartesian space which size is increasing proportional to the distance to the fixation point. This implies

that regions more distant to the pole are blurred stronger and therefore generate weaker edges. The edge detection thereby imitates an aspect of the human visual system: The blurred vision outside the focus.

Having taken these benefits into consideration, the edge detection itself could be optimized heavily. On the one hand, many optimizations, like the non-maximum suppression or the Savitzky-Golay filter, for generating an more accurate edge map were neglected due to the error-robustness of the grab cut algorithm. On the other hand, the texture gradient computation was removed completely by exploiting the advantages of the log-polar space. Additionally, the treatment of textured regions is supported by the selection of a larger kernel size for the edge detection such that internal edges get blurred even more.

The analysis of the results showed that the new algorithm is capable of segmenting various objects in simple environments almost without a loss of quality compared to the original approach by [Mishra et al., 2009]. The fixation-based approach is adequate way to rephrase the general segmentation problem, even though it has its limit: The graph cut and grab cut algorithm suffer from the so called 'short cut' problem in Cartesian space. Transferred to polar space, a similar effect occurs on very thin or elongated objects which results in cut off areas. This refers to a general challenge of this segmentation approach in polar space: Finding the 'right' mixing parameters for the regional and boundary term used in the grab cut algorithm. The analysis of the algorithm showed that universal parameters do not exist in the context of this algorithm. However, there are possibilities to avoid this problem. For example, this can achieved by selecting multiple fixation points on the object of interest, whereas the resulting segmentations get merged. This would be a interesting starting point for further investigation.

A related challenge is to find the 'correct' parameter for combining the computed color gradient and the depth gradient during the edge detection. On the one hand, this parameter depends strongly on the quality of the disparity map used to compute the depth gradient. On the other hand, the analysis showed, aside from the quality of the disparity map, that it is not possible to find a universal

parameter setting neither, mainly caused by the missing ‘contact boundaries’ in the disparity map. The implementation of an algorithm dealing explicitly with ‘contact boundaries’ would be a major improvement which should be explored in the future.

Finally, an analysis of the performance of the proposed algorithm has been done. It clearly showed that the efficiency of the new approach with the applied optimizations increased tremendously. Especially the improvements on the edge detection led to a runtime about 16 times faster than the original approach. The overall performance increased by a factor of seven. In conclusion, the implemented enhancements led to a new algorithm which is capable of segmenting a region of interest in a similar time as a human observer would need to identify an unknown object in a scene.

In summary, the algorithm implemented in the present thesis achieved the objectives formulated at the beginning in a very satisfactory way. Moreover, it offers a wide range of further optimization and extension options: As the algorithm only consumes one-fifth of the available CPU time, it could be implemented as a multithreading application in order to further increase the efficiency of the processor time. Major parts of the algorithm could even be computed on the GPU. This would boost the computational performance greatly, hence allowing a segmentation approach with multiple fixation points in real-time. As the segmentation is context-sensitive due to the selection of the fixation point, this approach could be utilized to create a hierarchical segmentation process by enabling the automatic choice of additional fixation points close to the initial one. This would, on the one hand, lead to a more precise segmentation of complex objects. On the other hand, hierarchical structures, like a window in a wall, could be detected. These are only a few application possibilities imaginable in the future.

Bibliography

[Adams, 2011] Adams, C. (2011). Leonhard euler and the seven bridges of königsberg. *The Mathematical Intelligencer*, 33(4):18–20.

[Adobe Systems Incorporated, 2012] Adobe Systems Incorporated (2012). *Adobe Photoshop CS6 User Guide*. Adobe Systems Incorporated.

[Alan and II, 2007] Alan, R. and II, P. (2007). On the computation of the discrete log-polar transform. *Transform*, pages 1–17.

[Araujo and Dias, 1997] Araujo, H. and Dias, J. M. (1997). An introduction to the log-polar mapping. *Proceedings II Workshop on Cybernetic Vision*, 0(1):139–144.

[Arbelaez et al., 2011] Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916.

[Arfken and Weber, 1985] Arfken, G. and Weber, H. J. (1985). The convolution theorem. *Mathematical Methods for Physicists*, pages 810–814.

[Belongie and Malik, 2000] Belongie, S. and Malik, J. (2000). Matching with shape contexts. In *IEEE Workshop on Content-based Access of Image and Video Libraries*, pages 20–26.

[Bernardino and Santos-Victor, 1999] Bernardino, A. and Santos-Victor, J. (1999). Binocular tracking: integrating perception and control. *IEEE Transactions on Robotics and Automation*, 15(6):1080–1094.

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*, volume 16. Springer, 1st ed. 2006. corr. 2nd printing edition.

[Boukala et al., 2003] Boukala, N., Rugna, J. D., and Colantoni, P. (2003). Hybrid color spaces applied to image database. *Proceedings of SPIE*, 5304(33):254–263.

[Boykov and Jolly, 2001] Boykov, Y. Y. and Jolly, M. P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Eighth IEEE International Conference on Computer Vision*, volume 1, pages 105–112.

[Bracewell, 1999] Bracewell, R. (1999). *The Fourier Transform & Its Applications*, chapter Heaviside's Unit Step Function, pages 61–65. McGraw-Hill Science/Engineering/Math, 3 edition.

[Bradski and Kaehler, 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1st edition.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 8(6):679–698.

[Cheng et al., 2001] Cheng, H. D., Jiang, X. H., Sun, Y., and Wang, J. (2001). Color image segmentation: Advances and prospects. *Pattern Recognition*, 34(12):2259–2281.

[Coifman, 1998] Coifman, B. (1998). A real-time computer vision system for vehicle tracking and traffic surveillance. *Tramsportation Research Part C: Emerging Technologies*, 6(4):271–288.

[Colantoni, 2004] Colantoni, P. (2004). Color space transformations.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

[Edmonds, 1970] Edmonds, J. (1970). *Submodular functions, matroids, and certain polyhedra*. New York: Gordon and Breach.

[Fairchild, 2005] Fairchild, M. D. (2005). *Color Appearance Models*. Wiley-IS&T Series in Imaging Science and Technology, Chichester, UK, second edition edition.

[Ford and Fulkerson, 1956] Ford, L. R. and Fulkerson, D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8(1):399–404.

[Ford and Fulkerson, 1957] Ford, L. R. and Fulkerson, D. R. (1957). A simple algorithm for finding maximal network flows and an application to the hitchcock problem. *Canadian Journal of Mathematics*, 09:210–218.

[Ford and Fulkerson, 1962] Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press.

[GIMP Documentation Team, 2010] GIMP Documentation Team (2010). *GIMP Documentation*.

[Greig et al., 1989] Greig, D. M., Porteous, B. T., and Seheult, A. H. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 51:271—-279.

[Guild, 1932] Guild, J. (1932). The colorimetric properties of the spectrum. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 230(681-693):149–187.

[Hammersley and Clifford, 1971] Hammersley, J. M. and Clifford, P. (1971). Markov fields on finite graphs and lattices. *Unpublished manuscript*, 3.

[Itti et al., 1998] Itti, L., Koch, C., and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259.

[Jonas et al., 1992] Jonas, J. B., Schneider, U., and Naumann, G. O. H. (1992). Count and density of human retinal photoreceptors. *Graefe's Archive for Clinical and Experimental Ophthalmology*, 230:505–510.

[Kanungo et al., 2002] Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. (2002). An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892.

[Kindermann and Snell, 1980] Kindermann, R. and Snell, J. L. (1980). Markov random fields and their applications. *Science*, 1(211739):142.

[Kolmogorov and Zabih, 2004] Kolmogorov, V. and Zabih, R. (2004). What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159.

[Kyrki and Kragic, 2011] Kyrki, V. and Kragic, D. (2011). Computer and robot vision. *Robotics & Automation Magazine*, 18(2):121–122.

[Lim et al., 2009] Lim, J. J., Arbelaez, P., and Malik, J. (2009). Recognition using regions. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1030–1037.

[Lindeberg, 1993] Lindeberg, T. (1993). Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention. *International Journal of Computer Vision*, 11(3):283–318.

[Littmann and Ritter, 1997] Littmann, E. and Ritter, H. (1997). Adaptive color segmentation - a comparison of neural and statistical methods. *IEEE Transactions on Neural Networks*, 8(1):175–185.

[Lucchese et al., 2001] Lucchese, L., Mitra, S. K., and Barbara, S. (2001). Color image segmentation : A state-of-the-art survey. *Citeseer*, 67(2):207–221.

[Malisiewicz and Efros, 2008] Malisiewicz, T. and Efros, A. A. (2008). Recognition by association via learning per-exemplar distances. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:1–8.

[Marr and Hildreth, 1980] Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 207(1167):187–217.

[Martin et al., 2004] Martin, D. R., Fowlkes, C. C., and Malik, J. (2004). Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549.

[Mishra, 2010] Mishra, A. K. (2010). *A Fixation Based Segmentation Framework*. PhD thesis, National University of Singapore.

[Mishra and Aloimonos, 2011] Mishra, A. K. and Aloimonos, Y. (2011). Visual segmentation of simple objects for robots. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA.

[Mishra et al., 2009] Mishra, A. K., Aloimonos, Y., and Cheong, L. F. (2009). Active segmentation with fixation. In *ICCV'09*, pages 468–475.

[Mishra et al., 2012] Mishra, A. K., Aloimonos, Y., Cheong, L. F., and Kassim, A. (2012). Active segmentation with fixation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):639–653.

[Mortensen and Barrett, 1995] Mortensen, E. N. and Barrett, W. A. (1995). Intelligent scissors for image composition. In *SIGGRAPH Proceedings In Computer Graphics*, pages 191–198.

[Mortensen and Barrett, 1998] Mortensen, E. N. and Barrett, W. A. (1998). Interactive segmentation with intelligent scissors. In *Graphical Models and Image Processing*, pages 349–384.

[Nadernejad and Sharifzadeh, 2008] Nadernejad, E. and Sharifzadeh, S. (2008). Edge detection techniques : Evaluations and comparisons. *Applied Mathematical Sciences*, 2(31):1507–1520.

[Peng and Veksler, 2008] Peng, B. and Veksler, O. (2008). Parameter selection for graph cut based image segmentation. *British Machine Vision Conference*, pages 160–170.

[Pham et al., 2000] Pham, D. L., Xu, C., and Prince, J. L. (2000). Current methods in medical image segmentation. *Annual review of biomedical engineering*, 2(1):315–337.

[Rayner, 1998] Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, 124(3):372–422.

[Reynolds, 2008] Reynolds, D. (2008). Gaussian mixture models. *Digital Signal Processing*, 45(2):1–5.

[Rother et al., 2004] Rother, C., Kolmogorov, V., and Blake, A. (2004). Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23:309–314.

[Scharr, 2000] Scharr, H. (2000). *Optimale Operatoren in der Digitalen Bildverarbeitung*. PhD thesis, Universitätsbibliothek.

[Schwartz and Greve, 1995] Schwartz, E. and Greve, D. (1995). Space-variant active vision: definition, overview and examples. *Neural Networks*, 8(7):1297–1308.

[Schwartz, 1984] Schwartz, E. L. (1984). Anatomical and physiological correlates of visual computation from striate to infero-temporal cortex. *IEEE Transaction on Systems, Man and Cybernetics*, SMC-14(2):257–271.

[Shapiro and Stockman, 2001] Shapiro, L. G. and Stockman, G. C. (2001). *Computer Vision*. Prentice Hall, New Jersey.

[Shmueli, 2007] Shmueli, A. (2007). Image segmentation using 1d matching: a combined segmentation and editing tool. Master's thesis, Tel-Aviv University.

[Suzuki and Be, 1985] Suzuki, S. and Be, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46.

[Tistarelli and Sandini, 1993] Tistarelli, M. and Sandini, G. (1993). On the advantages of polar and log-polar mapping for direct estimation of time-to-impact from optical flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):401–410.

[Vandenbroucke et al., 2003] Vandenbroucke, N., Macaire, L., and Postaire, J.-G. (2003). Color image segmentation by pixel classification in an adapted hybrid color space. application to soccer image analysis. *Computer Vision and Image Understanding*, 90(2):190–216.

[Viola and Jones, 2004] Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154.

[Wright, 1929] Wright, W. D. (1929). A re-determination of the trichromatic coefficients of the spectral colours. *Transactions of the Optical Society*, 30(4):141–164.

[Yilmaz et al., 2006] Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking: A survey. *ACM Computing Surveys*, 38(4).

DECLARATION OF ACADEMIC HONESTY

I hereby declare to have written this Master Thesis on my own. All parts of this assignment which are cited literally or in a rough summary from publications or other secondary material are recognizable, and I have clearly defined them with their respective references.