Freie Universität Berlin

Stochastic Object Pose Calculation for a Space Robot Manipulator

Master's Thesis

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science
in Computer Science

Submitted on August 23, 2014 by

Irena Kpogbezan-Lämmerer

Committee

Prof. Dr.-Ing. Raul Rojas | Dr. Hamid Reza Moballegh

Department of Mathematics and Computer Science

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Irena Kpogbezan-Lämmerer, Berlin, August 23, 2014

Acknowledgment

During this exiting project, I have learned a lot of interesting things and had the opportunity to gain insight into other fields of study. I want to thank Professor Rojas, Dr. Hamid Moballegh as well as the complete Berlin Rockets team, and the DLR for giving me the possibility to be a part of the project. The knowledge and experience of Dr. Moballegh as my supervisor have been of great value and I am thankful for his advice and constructive criticism.

Besides the people that have been directly involved in the project, I am also deeply grateful to my friends, my family and my partner, who have always supported me but have been especially encouraging and considerate in the last six months. I want to thank them for keeping my spirits and my confidence up and giving me valuable advice. This thesis and I would not be the same without them.

Irena Kpogbezan-Lämmerer

Contents

Abstract 5

List of Figures 7

List of Tables 8

List of Algorithms 8

List of Abbreviations 9

List of Symbols 10

1. Introduction 11
- 1.1. SpaceBot Cup . . . 11
  - 1.1.1. The challenge . . . 11
  - 1.1.2. The objects . . . 12
- 1.2. Hardware aspects of the robot . . . 12
  - 1.2.1. Sensors . . . 12
  - 1.2.2. Actuators . . . 14
  - 1.2.3. Mechanical structure . . . 14
- 1.3. State of the art . . . 16
- 1.4. Structure of this thesis . . . 17

2. Fundamentals 18
- 2.1. Robot Operating System . . . 18
  - 2.1.1. Framework structure . . . 18
  - 2.1.2. Important packages and tools . . . 19
- 2.2. Particle filters . . . 20
  - 2.2.1. Hidden Markov model . . . 21
  - 2.2.2. Recursive Bayesian filtering . . . 21
  - 2.2.3. Sequential Monte Carlo methods . . . 22
  - 2.2.4. Sequential importance resampling . . . 22
  - 2.2.5. Particle filtering . . . 23
  - 2.2.6. Discussion of particle filters . . . 24
- 2.3. HSV color space . . . 24
- 2.4. Pinhole camera model . . . 25

3. Estimation of orientation and position of the object 26
- 3.1. Initialization . . . 26
  - 3.1.1. Object finder . . . 27
  - 3.1.2. Number of particles . . . 27

3.2. Observation . . . 28
3.2.1. Object models . . . 28
3.2.2. Evaluation . . . 29
3.3. Resampling and prediction . . . 32
3.4. Implementation details . . . 33

4. Experiments **35**
4.1. Test set-up and execution . . . 35
4.1.1. Rigid camera . . . 36
4.1.2. Moving camera . . . 36
4.2. Results . . . 37
4.2.1. Object poses . . . 37
4.2.2. Object types . . . 37
4.2.3. Number of particles . . . 38
4.2.4. Vicinity . . . 39
4.2.5. Occlusion . . . 39
4.2.6. Moving camera . . . 40
4.2.7. Run time . . . 40
4.3. Discussion . . . 43

5. Recapitulation **44**

Bibliography **45**

A. Algorithms **47**
A.1. Resampling . . . 47

B. Experiment details **48**
B.1. Test cases . . . 48
B.2. Object poses . . . 51

C. Experiment results **52**
C.1. Object poses . . . 52
C.2. Object types . . . 54
C.3. Number of particles . . . 56
C.4. Vicinity . . . 58
C.5. Occlusion . . . 60

Abstract

Exploring other planetary surfaces with autonomous robots is a very complex task and requires a lot of research. To give an impulse, the DLR organized the first SpaceBot Cup in 2013. The team Berlin Rockets from the Freie Universität constructed a six-legged robot to take part in the competition. This thesis presents an approach for the stochastic estimation of an object's pose. The estimate will be used by the robot to grasp the object. The estimation is done with a particle filter that uses depth and color data from the robot's Kinect for Windows™. The object models consist of 3D point that are evenly distributed over the object's surface. Additionally, a few specific pixels from the vicinity of the object are also taken into account to reduce the rate of false positives. To test the particle filter, several test cases were conducted. The accuracy of the results is unfortunately very low, which shows that the measures taken did not have the desired effect.

List of Figures

1.1. Object specifications . . . 12
1.2. Depth perception technique of the Kinect . . . 13
1.3. Prototype design . . . 14
1.4. Schematic representation of the arms . . . 15
1.5. Gripper for the cup . . . 16

2.1. Example of a tf tree . . . 19
2.2. Screenshot of rviz . . . 20
2.3. Hidden Markov model . . . 21
2.4. Pinhole camera model . . . 25

3.1. Canny edge filter applied to image of planetary surface . . . 28
3.2. Models of the objects . . . 29
3.3. Evaluation of the vicinity . . . 31
3.4. Assigning importance weights to particles . . . 33

4.1. Test scene . . . 35
4.2. Error for different poses . . . 37
4.3. Error for different object types . . . 38
4.4. Error for different upper bounds for the number of particles . . . 38
4.5. Error for consideration/ no consideration of the vicinity . . . 39
4.6. Error for occluded objects/non-occluded objects . . . 39
4.7. Screenshots from the experiments for the moving objects . . . 41
4.8. Average duration per iteration . . . 42

B.1. Object poses . . . 51

C.1. Error in position for different poses . . . 52
C.2. Error in orientation for different object poses . . . 53
C.3. Error in position for different object types . . . 54
C.4. Error in orientation for different object types . . . 55
C.5. Error in position for different upper bounds of particles . . . 56
C.6. Error in orientation for different upper bounds of particles . . . 57
C.7. Error in position for consideration /no consideration of the vicinity . . . 58
C.8. Error in orientation for consideration /no consideration of the vicinity . . . 59
C.9. Error in position for occlusion/no occlusion of the objects . . . 60
C.10. Error in orientation for occlusion/no occlusion of the objects . . . 61

List of Tables

B.1. Test cases 48
B.2. Poses of the battery 51
B.3. Poses of the cup 51

List of Algorithms

1. Resampling 47

List of Abbreviations

3D	3-dimensional
2D	2-dimensional
DBN	dynamic Bayesian network
DLR	Deutsches Zentrum für Luft- und Raumfahrt
HMM	hidden Markov model
HSV	hue, saturation, value
ID	identifier
IS	Importance sampling
IMU	inertial measurement unit
RGB	red, green, blue
RGB-D	red, green, blue, depth
ROS	Robot Operating System
SIR	Sampling importance resampling
SIS	Sequential importance sampling
SMC	Sequential Monte Carlo

List of Symbols

h_t	hidden state $h \in \mathbb{R}^n$ of the system at time step t.
o_t	observation $o \in \mathbb{R}^m$ emitted by the system at time step t, $m \leq n$.
$o_{1:t}$	$\{o_1, o_2, \ldots, o_{t-1}, o_t\}$.
$w(h)$	importance weight $w \in \mathbb{R}$ of h.
$p(h)$	probability distribution p of h.
$q(h)$	proposal distribution q of h, importance density.
f	focal length of the camera.
p_n	particle n, with $1 < n < N$.
h_n	hidden state h represented by of particle p_n.
$b(h_n)$	belief of hypothesis h of particle p_n.
M	number of grid points of an object that are projected into the image.
V	number of vicinity pixels of an object that are projected into the image.
g	grid point.
$x(g)$	$x(g) \in \mathbb{R}$, x-coordinate of grid point g.
$y(g)$	$y(g) \in \mathbb{R}$, y-coordinate of grid point g.
$z(g)$	$z(g) \in \mathbb{R}$, z-coordinate of grid point g.
C	binary result of color segmentation from RGB image.
D	depth image.
s_C	$s_C \in [0, 1]$, color score.
s_D	$s_D \in [0, 1]$, depth score.
$s_{V,C}$	$s_{V,C} \in [0, 1]$, color score of the vicinity.
$s_{V,D}$	$s_{V,D} \in [0, 1]$, depth score of the vicinity.

1. Introduction

Humankind has been dreaming of visiting and even living on other planets for a long time. The technological advancements in the last decades have brought the realization of that vision much closer. Yet, with the current state of the art in science and technology, it is still a futuristic dream. To make it safe for humans to step foot on another world, that planet has to be extensively studied first. However, satellites can only provide information from afar by orbiting the planet. To have a closer look and gather more information, autonomous robots for exploring the planetary surface are needed.

Autonomous robots have to be able to perceive their environment, to evaluate those perceptions and to make decisions based on those evaluations. They need to explore their environment, navigate to destinations of interest, avoid obstacles like rocks or sand, examine their vicinity for research relevant objects and possibly grasp and transport these objects to a bas station.

The objective of this thesis is the processing of the data from the sensors of a robot that was built for the exploration of planetary surfaces. The thesis presents the implementation of a particle filter that uses the data from a Microsoft Kinect for Windows™ to estimate the pose of an object in close proximity to the robot in order to enable the it to grasp the object properly and transport it to a base station. The robot was developed in the context of the SpaceBot Cup, a student competition which was held and organized by the Deutsches Zentrum für Luft- und Raumfahrt (DLR) for the first time in 2013.

1.1. SpaceBot Cup

This section shortly explains the SpaceBot Cup and its rules to clarify the conditions under which the particle filter is applied.

1.1.1. The challenge

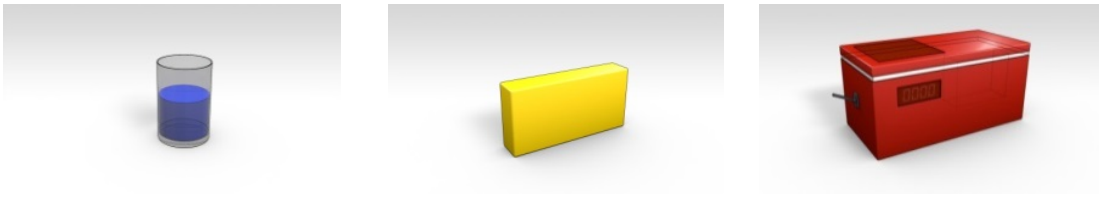
Each team has to develop a robot that is able to autonomously explore the surface of other planets. The robot has to complete as many tasks as possible within one hour. The robot is positioned in a field that simulates a planetary surface. The field is 28 m $\times$ 36 m in size and is designed with hills of various heights and inclines, a ditch and fields of sand, gravel and even small boulders.

The robot is initially given a rough map of the field, but has to gather as much additional information as possible about its environment and compile a detailed map, containing the positions of obstacles or the inclination of a hill. To fulfil that task, the robot has to navigate between or overcome the obstacles in the field. While navigating through the terrain, the robot has to look for two different objects and a base station. The objects have to be grasped and transported to the base station. Each one of the objects has to be mounted on the base station in a specific way, depending on the type of the object.

To further simulate the conditions of an exploration mission on another planet, the teams are divided into two groups. The first group is spatially separated from the field, i.e. in another room. They can send commands to the robot and receive data from it, although with a 2 second delay in each direction. The second group is allowed to stand next to the field and to intervene in case of an emergency like a failed motor or the robot being stuck in sand. Each sent command or intervention from either of the two groups results in a time penalty. The two groups are not allowed to communicate directly with each other.

1.1.2. The objects

The objects that the robot has to find have different colors, shapes and dimensions. The first object is a blue, cylinder shaped cup (figure 1.1a). It is hollow, open on the top and contains distilled water. The second object is a yellow cuboid representing a battery pack (figure 1.1b). The third and last object is the base station, a red cuboid with bigger dimensions than the battery (figure 1.1c). The base station additionally has a scale on top, a slot for the battery on one side and a small lever on a different side. The exact dimensions of all objects are listed in figure 1.1.



(a) Cup
size: ca. 12 cm x 8 cm x 8 cm
weight: max. 800 g

(b) Battery
size: ca. 10 cm x 20 cm x 4 cm
weight: ca. 800 g

(c) Base station
size: ca. 30 cm x 40 cm x 20 cm
weight: not given

Figure 1.1.: Object specifications [4]

Once transported to the base station, the cup is supposed to be placed on the scale on top of the base station to test how much water was lost during transportation. The battery shall be put into the slot on the side of the base station. After mounting the objects at the base station, the lever is to be turned. The lever activates a small lamp that indicates the level of success.

1.2. Hardware aspects of the robot

This section introduces the robot itself, focussing on the sensor that was used as the main data source and on the concept for the arms for grasping the objects.

1.2.1. Sensors

Microsoft Kinect for Windows™

The particle filter uses the data from the Kinect for its estimation. The Kinect is a RGB-D camera, developed by Microsoft in cooperation with PrimeSense.[14] It was originally intended for controller-less

video games. However, due to its comparably high quality data and low price, the scientific community quickly discovered its value for robotic applications.[11]

The Kinect has two modes, a default mode and a near mode. In the default mode, the viewing range of the depth camera is 0.8 m to 4 m. In near mode, the viewing range of the depth camera is 0.4 m to 3 m.[15] The particle filter is only used when the robot has already found an object and has navigated to it, so that the object will be close to the robot. Therefore, in this thesis the Kinect is set to near mode.

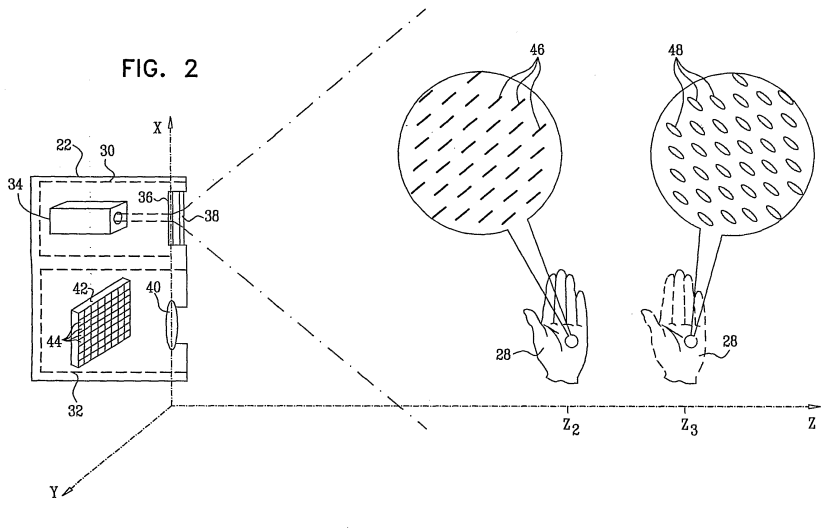


Figure 1.2.: Illustration of the depth perception technique of the Kinect. Image from [9]

The Kinect uses structured light in combination with an astigmatic optical element for depth perception.[9, 11] Structured light is a technique where a light source emits a known pattern that falls onto the objects in front of it. An illustration of that method is shown in figure 1.2. A light source (34), which is positioned at a fix distance from the camera, emits infrared light which is filtered by a transparency (36) creating the pattern. The pattern is projected onto an object (28) and thereby distorted. The different orientations of the ellipses are caused by an astigmatic optical element (38). The distortion of the pattern is captured by a camera (32). Shape, size and orientation of the ellipses can be used to compute the distance of the object.[9]

Velodyne HDL-32E laser scanner

This laser scanner has 32 laser beams with vertical angle of $+10°$ (up) to $-30°$ (down), is able to rotate $360°$ and has an optical range of $70m$ while giving an accuracy of less than $2cm$. Each scan contains around 70 000 points.[28] It is used to scan the field and gain information about the environment before starting to explore.

Inertial measurement unit

An inertial measurement unit (IMU) is a device typically containing gyroscopes, accelerometers or magnetometers. They are used for gathering data about the velocity or orientation of robots or other machines.

1.2.2. Actuators

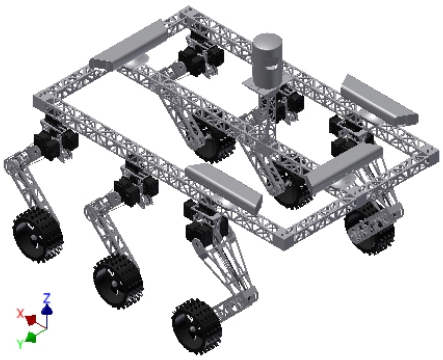
Each leg uses 6 motors by the company Dynamixel (2 RX-64 motors and 4 RX-28 motors). The motors are brush-commutated and have hall sensors for contactless position estimation.

1.2.3. Mechanical structure

The mechanical structure of the main body of the robot was designed by Dr. Hamid Moballegh and Dipl.Ing. Jan Streckenbach, while the author of this thesis was responsible for the design and construction of the arms. The body of the robot is made out of aluminium profiles with half-timber cut-outs.

Locomotion system

The robot has six legs. Each leg is equipped with a wheel. The idea behind this design is to enable the robot to walk as well as drive, depending on the terrain. Another crucial part of the design is the placement of the motors. All motors are hinged to the upper part of the leg, in order to prevent dust or grit getting easily into the motors. The wheels can be turned, so that the robot can drive sideways. Early prototypes of the robot and the legs are pictured in figure 1.3a and 1.3b.



(a) Early design for a prototype of the robot

(b) Early design of a prototype of a leg

Figure 1.3.: Early Prototypes. Pictures by Jan Streckenbach

Structure of the arm

If the robot has found and grasped an object and is on its way to the base station, it is not unlikely that it will locate the second object on the way. In that case there are two possible solutions. The first solution is to transport each object separately and driving to the base station multiple times. The second solution is to transport both objects simultaneously and drive to the base station only once. Since each trip to any destination requires finding and navigating a path that is not too long but avoids obstacles is very complex, it would be ideal to minimize the amount of necessary trips. Thus, this robot was designed to have two arms.

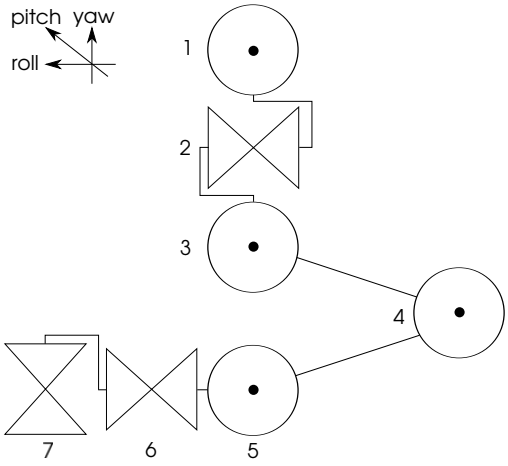


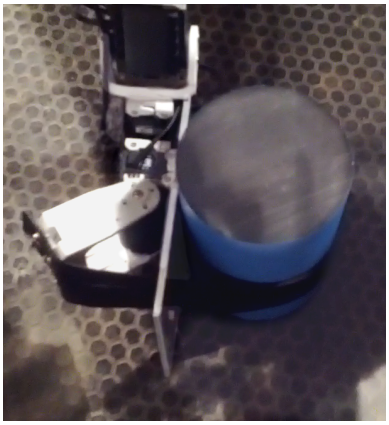
Figure 1.4.: Schematic representation of the arms.

Both arms have the same basic design, which is depicted in figure 1.4. The numbers are the ID of the joint next to them. The various shapes signify the orientation of the joint; joints 1,3,4 and 5 rotate around the pitch axis, joints 2 and 6 rotate around the roll axis and joint 7 rotates around the yaw axis. The joints with the IDs 1 to 3 compose the shoulder and the joints with the IDs 5 to 7 compose the gripper. Joint 4 is the arm's elbow. Joint 1 is mounted at the front of the frame at which the legs are mounted as well. The shoulder only has two degrees of freedom, but the missing degree of freedom substituted by using the legs.

The two arms differ in the design of their gripper. Each arm has a gripper that is specifically designed to grasp and carry one type of object. The arm for the cup is equipped with a sling that will be positioned directly above the cup, let down and tightened around it, so that it is pressed against a metal plate that is mounted at joint 7 (see figure 1.5). The arm for the battery is equipped with a vacuum exhauster that will be positioned on the side of the battery. Initially and during transport, both arms will be contracted, as in figure 1.4, and the objects will remain in the grippers until mounting at the base station.

For the transport of the cup, it is essential that it is kept upright to minimize the loss of water. Since the cup is quite heavy and the motors are not strong enough, the arm needs additional support for carrying the cup. For this purpose, there is an extra motor mounted to a pole on the main frame. Attached to this motor is a coil with an elastic line whose other end is attached to joint 5. If the motor detects a force pulling at the line, it revolves backwards and thereby coils up the line. This has the effect of pulling up the arm and thus supporting the grasping of the cup.

(a) The sling of the gripper is positioned directly above the cup and is ready to lower itself, so that it encloses the cup.



(b) The sling has been tightened after enclosing the cup, so that the cup is pressed against a metal plate. It's now ready to be pulled up.

Figure 1.5.: Gripper for the cup in consecutive stages.

1.3. State of the art

Pose estimation of objects and object tracking are often accomplished by the help of particle filters. The foundation for particle filters was built as early as 1993 by Gordon et al. [10]. They developed an algorithm that represented the posterior distribution using randomly picked samples that were iteratively updated and propagated based on Bayesian filtering and that could work with non-linear models and non-Gaussian noise. Isard and Blake [12] used the sampling strategy from Gordon et al. [10] to track the position of faces in image streams. They used curves to model the contours of a head and looked for features along those curves. This was one of the first visual application of particle filters.
Pupilli and Calway [22] extended the approach of Isard and Blake [12] to tracking 3-dimensional objects. Their object models are defined by the geometrical 3D edges of the objects. To improve the edge matching, Pupilli and Calway grouped line segments to junctions instead of matching single edges. Due to computational limits, they were limited to geometrically simple and not self-occluding objects. Klein and Murray [13] made it possible to track more complex and even self-occluding objects in 3D by utilizing the computational power of modern graphic cards. Expensive operations like hidden line removal were simply out-sourced to the graphics hardware.

A great variety of particle filters exists for object tracking applications. Mörwald et al. [16] – like Klein and Murray [13] – uses the computational power of graphics card. They not only use the geometric edges of an object but also edges from the texture, which improves the performance in case of occlusion.
While edge matching is very common in object tracking applications, there are also some approaches relying on color information. Nummiaro et al. [18] model their objects using a weighted histogram of colors. Pixels near the edges are weighted less to account for blurring. Their algorithm thereby also evaluates shape to a small extent. To deal with illumination changes, the models are updated iteratively. Taiana et al. [27] also rely on color information, but instead of modelling the complete object, they only draw random pixels from within and outside the object's projected 2D surfaces and evaluate their color distributions. Pixel operations are computationally very cheap, thus their algorithm is very efficient.

There have also been various efforts that do not use particle filters, most of which are rather problems of efficient searching in data structures storing the various object models. Olson and Huttenlocher [19]

formulate the problem as a problem of an efficient search algorithm. The authors model their objects as edge maps and cluster the edge pixels by orientation. They combine a different 2D views of one object into a set and use that set as a 3D model of that object. Similar to Nummiaro et al. [18], Ekvall et al. [7] use color histograms. But unlike Nummiaro et al. [18], they employ color co-occurrence histograms, i.e. rather than aggregating over the entire image, their histograms contain information about color distributions in rows and columns, thereby holding information about geometric structures of the objects. Collet et al. [3] iteratively cluster image features by spacial proximity, assuming that close-by features likely belong to the same object. After constructing the clusters, they try to find matches between the features of the model and the features of each cluster. Reinbacher et al. [23] create reference views by placing virtual cameras on a sphere around a CAD model of the object. The reference views only regard the silhouettes of the objects, including the orientation of edges. The views are then grouped by similarity and stored in a hierarchical structure which can then be queried in a top-down manner.

1.4. Structure of this thesis

The thesis is structured as follows: the second chapter gives an overview about important fundamentals such as the software framework that was used and the concept of particle filtering algorithms. The third chapter explains in detail how the problem of calculating the pose and orientation of an object using the sensor data from the Kinect was approached. The fourth chapter evaluates the solution from chapter three. It starts with a description of the set-up of the tests, followed by the results and a discussion thereof. The fifth and last chapter gives a short summary of this thesis and an outlook into possible future work.

2. Fundamentals

The following chapter gives an overview about relevant components and theoretical concepts. The first section introduces the employed software framework. The next section shortly explains the fundamentals of particle filters. The other sections introduce two concepts that have been important for the image processing.

2.1. Robot Operating System

Robot Operating System (ROS) is an open source software framework for building robot applications under the BSD license. Its usage is very widespread: "Already firmly established among the research community, ROS is now set to become the industry standard." [8] The framework provides a hardware abstraction from the robot for software development. The developers of ROS designed a highly modularized structure for the framework so that users are able to combine packages and tools as needed. As an open source project, ROS allows everyone to contribute new code. The list of packages is therefore constantly expanding. It lists packages for self-localization, for processing point clouds, for converting between ROS image messages and OpenCV[1] images, for visualizing data and many more.[24] Apart from providing these packages, ROS also allows the integration of important open source libraries, such as OpenCV [20] for image processing and PCL [21] for the processing of point clouds. Another crucial benefit of ROS is that it is equipped with many hardware drivers, for example for the Kinect and the Velodyne Laser Scanner.

2.1.1. Framework structure

Packages consist of small, independent programs called nodes. Nodes communicate by publishing own topics or subscribing to topics published by other nodes. Topics are buses that are addressed by a specific name and are assigned a specific message type. That message type can either be one of the many predefined types by ROS or a new type defined by the developer. The publishing node first has to register its topic in the system, name it and define its message type. The subscriber nodes then regularly retrieve the new messages from the buses.

Another method of communication in ROS are services. While the publisher/subscriber method provides broadcasting functionality, services allow direct communication between two nodes.

[1] OpenCV is a library for image processing tasks.

2.1.2. Important packages and tools

The following is a list of packages and tool provided by the ROS framework that were used in this thesis.

tf

The *tf* package stores a tree representation of all joints of the robot and their respective coordinates w.r.t. each other. An example is depicted in figure 2.1. Each joint of the robot is represented by its coordinate frame. The colors of the axes signify their dimension. Using that tree, the package can compute the coordinates of any point in the coordinate frame of any joint in relation to any other coordinate frame. The entries in the *tf* tree also contain time stamps so that the coordinates of a point can be computed for any time in the past.

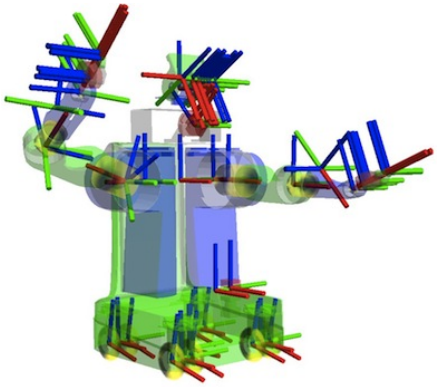


Figure 2.1.: Example of a tf tree. Image from [26].

rviz

rviz is a 3D visualization tool which can display RGB camera images, depth images, point clouds, but also self-defined message types and the *tf* tree. A screenshot of *rviz* with explanations is shown in figure 2.2. The top left frame shows the depth image from the Kinect, with a few brighter pixels that have been added by the particle filter. The bottom left frame shows the RGB image from the camera, also with manipulations by the program. The big frame on the right is for 3D visualizations such as self defined markers as pictured or point clouds. Also visualized are the coordinate frame (the red, green and blue lines) and a 3D grid for orientation. The small frame on the right is where all the other frames are configured. The add button adds new frames of various types (e.g. images, point clouds, coordinate frames). Once added, the topic containing the data to visualize can be specified.

sensor_msgs

This package defines a number of message types for the communication with sensors. There are message types for many different types of data, such as images, point clouds, or temperature data. The sensors can make their data available to the nodes by converting it into these message types.

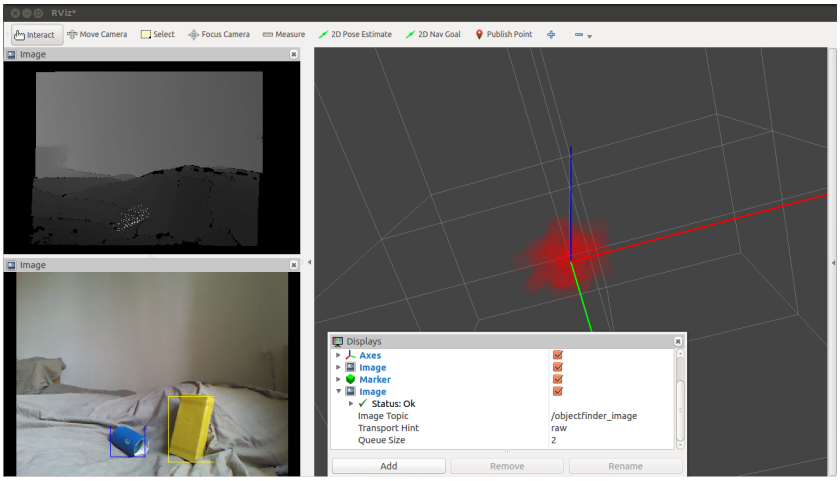


Figure 2.2.: Screenshot of rviz.

cv_bridge

When a node receives image data from a sensor, that data is formatted as some message type, e.g. one of the *sensor_msgs* message types. Often, that image data still has to be processed. OpenCV is an open source library that specializes on image processing. The main purpose of the *cv_bridge* package is to convert the image data from *sensor_msgs* into OpenCV-compatible data types, so the image data can be processed with OpenCV library functions.

visualization_msgs

The *visualization_msgs* package defines message types that can be used for visualizing self-defined data in *rviz*. The user can pick from cubes, cylinders, arrows and other shapes, define their dimensions, color and pose.

rosbag

rosbag is a tool that can record the messages in specific topics and store them in so called .bag files. The recorded messages can later be played back to their topics.

2.2. Particle filters

Particle filters are algorithms for state estimation. They are based on a number of other concepts. This section will explain those concepts and how they are related to the particle filter.

2.2.1. Hidden Markov model

Particle filters assume the process model to be a hidden Markov model (HMM) (cf. [12, 5]). A HMM is a Markov process where the states themselves are not directly measurable and therefore *hidden*, but emit observations. Hidden states are random variables that represent the current, non-measurable state of the system. The observations are measurable representations of the true state of the underlying system, e.g. an image or a temperature.

A (first-order) Markov process is any stochastic process for which the Markov property holds. The Markov property states that the next state of a variable only depends on its current state and not on any states from the past:

$$P(h_t|h_{t-1}, h_{t-2}, \ldots, h_2, h_1) = P(h_t|h_{t-1}), \tag{2.1}$$

where $h_t \in \mathbb{R}^n$ is the current state and h_{t-i} is the state from i steps into the past, with $t-1 > i > 1$. Applied to the HMM, the Markov property expresses that the current, observable measurement only depends on the current hidden state and that the current hidden state only depends on the previous hidden state. The dependencies in an HMM are illustrated in figure 2.3. The circles represent the random variables and their dependencies are indicated by the arrows. h_{t-1} and h_t are the hidden, unobservable states while o_{t-1} and o_t are the measurable observations, with t being the present time step.

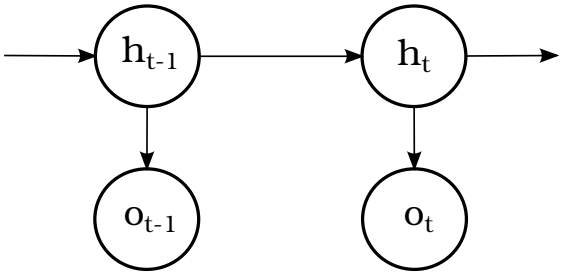


Figure 2.3.: Conditional dependencies between the variable states and the observations in a HMM.

2.2.2. Recursive Bayesian filtering

A dynamic Bayesian network (DBN) is a process model that represents the conditional dependencies of random values and their progression over time. This model also includes the HMM.[17, Ch. 2.3] By extension, the process model of a particle filter is a specific type of DBN. Therefore, particle filters are a type of algorithm that runs on a Bayesian network and can be counted to the Bayesian filtering algorithms (cf. [2])

Recursive Bayesian filtering aims to estimate the hidden state of a system h_t at time step t or even the complete state sequence $h_{1:t}$ given a sequence of observations $o_{1:t}$. The probability of the calculated state h_t is called belief b:

$$b(h_t) = p(h_t|o_{1:t}) \tag{2.2}$$

where $b(h_t)$ is the belief about the hidden state h_t and $o_{1:t}$ is the sequence of observations up to time step t: $\{o_1, o_2, \ldots, o_{t-1}, o_t\}$.

The foundation of the algorithm is Bayes rule:

$$P(h|o) = \frac{P(o|h) \cdot P(h)}{P(o)}. \tag{2.3}$$

Recursive Bayesian Filtering can be divided into two steps; namely the prediction step and the update step. In the prediction step the algorithm tries to estimate the probability distribution of the next state of the system $p(h_t)$, given all previous observations $o_{1:t-1}$:

$$p(h_t|o_{1:t-1}) = \int p(h_t|h_{t-1})p(h_{t-1}|o_{1:t-1})dh_{t-1}, \tag{2.4}$$

where $p(h_t|h_{t-1})$ models the state transition probability of the system and $p(h_{t-1}|o_{1:t-1})$ is the posterior distribution, i.e. the belief of the previous iteration. In the continuous case, the equation needs to be integrated over all possible predecessor states h_{t-1} (in the discrete case, the probabilities of all predecessor states are summed up rather than integrated.).

When a new observation is available, the algorithm can compute the new belief $p(h_t|o_{1:t})$ based on the last prediction $p(h_t|o_{1:t-1})$ and the new observation o_t:

$$p(h_t|o_{1:t}) = \frac{p(o_t|h_t)p(h_t|o_{1:t-1})}{p(o_t|o_{1:t-1})}, \tag{2.5}$$

where $p(o_t|h_t)$ models the likelihood of a sensor making observation o_t while the hidden state of the system is h_t, $p(h_t|o_{1:t-1})$ is the prior distribution of time step t and

$$p(o_t|o_{1:t-1}) = \int p(o_t|h_t)p(h_t|o_{1:t-1})dh_t, \tag{2.6}$$

is a normalizing term which – in the continuous case – needs to be integrated over all possible states h_t.

2.2.3. Sequential Monte Carlo methods

Monte Carlo methods are a set of methods where a set of independent random samples is used to give an estimation of a probability distribution. Sequential Monte Carlo (SMC) methods extend that concept to dynamic systems by reusing the previously drawn sample sets in consecutive iterations.

Particle filters use their set of particles to give a probability distribution of the system state. Due to the random element in choosing the particles, particle filters also belong to the group of SMC methods.

2.2.4. Sequential importance resampling

SMC methods approximate a distribution by randomly drawing samples from it. The goal is to have a set of samples that is a suitable representation of the distribution. Sampling importance resampling (SIR) is one of the many methods trying to achieve that. SIR is based on Importance sampling (IS), which is a method for taking samples from a distribution $p(h)$. The aim of IS is to increase the significance of the sample distribution. It is difficult to draw a good sample from a distribution, especially in high dimensional state spaces. If the distribution has long tails, IS will pick mostly samples that don't belong

to those parts. The algorithm is given an initial proposal distribution $q(h)$ consisting of samples, which are then given importance weights $w(h) \in \mathbb{R}$:

$$w(h_n) = \frac{p(h_n)}{q(h_n)}, \tag{2.7}$$

where $1 < n < N$ and N is the number of samples. After normalizing the weights, IS chooses the samples with higher weights.

Sequential importance sampling (SIS) extends this concept to dynamic systems. The samples of one iteration will be used as the proposal distribution for SIS in the next iteration. The weights are calculated by using the probability and proposal distributions of the previous iterations:

$$w(h_t) = \frac{p(h_1)p(h_2|h_1) \cdots p(h_t|h_1, h_2, \cdots, h_{t-1}, h_t)}{q_1(h_1)q_2(h_2|h_1) \cdots q_t(h_t|h_1, h_2, \cdots, h_{t-1}, h_t)}, \tag{2.8}$$

where q_t is the proposal distribution of time step t and is constructed sequentially:

$$q(h_{1:t}|o_{1:t}) = q(h_1) \prod_{i=2}^{t} q(h_t|h_{1:t-1}, o_{1:t}) \tag{2.9}$$

One disadvantage of SIS is the weight degeneracy problem. After a few iterations, most weights are quasi zero with only a few samples having significant weights, leading to a higher variance and thereby lower sample distribution quality. SIR is a variation of SIS that avoids this problem by adding an additional resampling step. In this step, the samples with low weights are replaced by copies of the samples with higher weights. This measure lowers the variance and improves the sample distribution quality.

2.2.5. Particle filtering

As Murphy puts it, "Particle filtering is just sequential importance sampling with resampling[...]."[17, p. 86] That is why they are often alternatively called SIR filters in the literature. Particle filters are algorithms for the estimation of the state of a dynamic system. Although they can be used for most problems, they are often used in computer vision applications such as self-localization of robots or object tracking.

Their system model is an HMM. In the previously mentioned types of applications, the hidden states would be either the position of the robot or the pose of an object while the observations would be sensor data from motors, IMUs or cameras.

The combination of recursive Bayesian filtering and SIR is a particle filter. It recursively estimates the current state based on an observation sequence using a prediction and an update step. The posterior distribution of the state is represented by a set of randomly picked samples. A resampling step is added to improve the sample distribution quality.

The randomly picked samples are called particles. They are tuples out of a point in the state space and an importance weight. As a first proposal distribution and prior for the first iteration, a set of particles is generated (equation (2.9) with $t = 1$), with equal weights. With the first incoming observation, the weights, or beliefs of each particle are updated (equation (2.5)). After the update step, the particles are resampled. Particles with low importance weights are deleted and particles with high importance weights are multiplied, so that the overall number of particles stays the same. The last step in the cycle tries to

predict the next state of the system (equation (2.4)), i.e. moves the particles in the state space according to the motion model (or more general, action model, if the application does not track movement). This new set of particles is the new proposal distribution $q2$.

2.2.6. Discussion of particle filters

Advantages

The most significant advantage of particle filters is their flexibility concerning the underlying models. In contrast to e.g. Kalman filters, particle filters can work with non-linear models, and non-Gaussian noise. That means that particle filters can be applied to all state estimation problems.

Also, since each particle has its own state hypothesis and weight and is independent from all other particles, particle filters can support multiple hypotheses simultaneously and thus estimate multi-modal distributions. This makes it much easier to recover from losing track of the object, because the development of a new hypothesis (i.e. a peak in the probability distribution) does not require the loss of another. Both hypotheses can co-exist. If the particle filter develops a hypothesis that turns out to be wrong, it can always return to the other hypotheses.

Another advantage is that the evaluation and weighting of the particles can easily be parallelized which can significantly speed up the computation.

Disadvantages

Since the probability distribution of the system state is represented by samples, the result can only be an approximation, whereas e.g. Bayesian filters compute the optimum. A higher number of particles can give a more accurate approximation, but since every particle needs to be weighed and evaluated in each iteration, the increasing of the number of particles also increases the computational cost. It is hard to determine the optimal number of particles.

2.3. HSV color space

HSV stands for hue, saturation, and value and is a system for describing color that is based on the human perception of color. The human eye has four distinct types of sensors. One type detects the level of brightness (rod cells), and each one of the other three (cone cells) each detects the amount of a specific color in light (red, green and blue, respectively). HSV also distinguishes between chroma (i.e. color) and luma (i.e. brightness).

A color in HSV is described by three parameters. The first parameter describes the hue, given as an angle between $0°$ and $360°$. This angle points to a certain hue in a color wheel. $0°$ is red, $120°$ is green and $240°$ is blue. The values in between are mixtures thereof. The hue value is equivalent to the ratio of cone cells in the human eye that react to a light ray. The second parameter is saturation and is given in percentage. This parameter is equivalent to the level of excitement of the cone cells. The third parameter is called value and is also given in percentage. It describes the level of brightness and is the equivalent to the rod cells of the human eye.

Since the concept of HSV is so closely related to the concept of the human eye, the description of colors in this color system is very intuitive for humans. It is – amongst various other applications – therefore often used for color segmentation.

2.4. Pinhole camera model

The pinhole camera model describes the relationship between the pose of an object in the real world and its position in a 2D image, when it is projected through a pinhole camera. Physically, a pinhole camera is simply a box with a small hole. If light falls through this hole, it falls onto the inner back wall of the box. This projects an image of the outside of the box onto the inner wall of the box. The back of the box becomes the image plane and the pinhole functions as a camera lens (see figure 2.4).

The 2-dimensional (2D) coordinates of a light ray from a 3-dimensional (3D) object in the image plane can be determined with the following equation:

$$\begin{pmatrix} y_{2D} \\ x_{2D} \end{pmatrix} = \frac{f}{z_{3D}} \begin{pmatrix} y_{3D} \\ x_{3D} \end{pmatrix}, \tag{2.10}$$

where x_{2D} and y_{2D} are the coordinates of the projected point and x_{3D},y_{3D} and z_{3D} are the coordinates of the 3D point. f is the focal length of the camera.

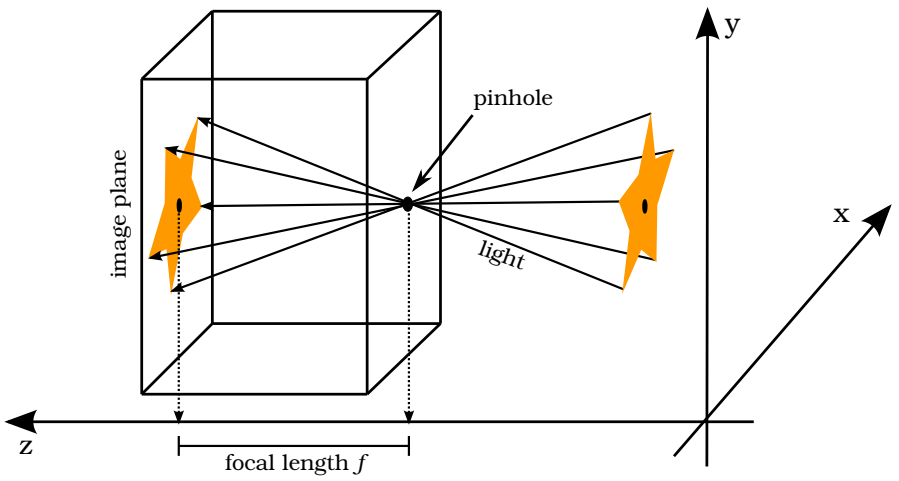


Figure 2.4.: Concept of a pinhole camera.

3. Estimation of orientation and position of the object

This chapter presents the approach that was taken in the implementation of the particle filter. As an introduction, the application scenario is specified. Subsequently, the chapter shows aspects of the initialization stage. The following section describes how each particle is evaluated based on the observations. Afterwards, the resampling step of the particle filter is explained. The chapter closes with the implementation details.

The scenario in which this implementation of a particle filter will be used is that while the robot is exploring the terrain. It has found one of the objects and has navigated towards it, now standing in close proximity to it. At this point, the robot has an approximate position of the object. The next step is to grasp it in order to transport it to the base station. To be able to properly grasp it, the robot needs a more accurate position and additionally the orientation of the object. It is the task of this implementation of a particle filter to provide an exact object pose, using the RGB and depth image data from the Kinect.

The particles are a representation of the state and consist of a hypothesis about the state, an importance weight and a belief:

$$p_n = \{h_n, w(h_n), b(h_n)\}, \tag{3.1}$$

where h_n is a point in the 6-dimensional state space. This point is the hypothesis about the hidden state of particle p_n. h_n itself consists of a vector describing the object's position in 3D space and a vector describing the object's orientation. $w(h_n)$ is the weight of the particle's hypothesis about the hidden state and $b(h_n)$ is the belief of the hypothesis of the particle. Both weight and belief are initially 0.

3.1. Initialization

With higher dimensionality, state spaces grow very fast, which makes it increasingly difficult to draw significant sample sets. Since the particles in this application have six degrees of freedom, the state space is large. To achieve better results, Nummiaro et al. strategically place their particles in regions with a higher probability of finding a good estimate.[18]With less space to cover, it is easier to find a representative sample set. The reduction of the search area allows a reduction of the number of necessary particles.

This particle filter works with a reduced state space as well. Particles are initialized at randomly chosen positions within the delimited search area and with randomly chosen orientations. The behavior node of the robot determines which type of object the particle filter should look for.

3.1.1. Object finder

The object finder is a ROS node written by Benjamin Aschenbrenner, also for the SpaceBot Cup project. It returns a rough estimate of the position of the object which the particle filter uses to reduce the state space.

The object finder uses the RGB color image from the Kinect. Due to HSV having significant advantages in color segmentation, it first converts the RGB image into an HSV image. It then performs the color segmentation for the respective object color, using the `inRange(...)` function from OpenCV. This function takes as parameters the input image, an output image and an upper and a lower bound for the object's color. For each pixel in the input image, the function tests whether its color value falls between the upper and the lower bound. The binary output image marks all pixels for which this is false with the value zero.

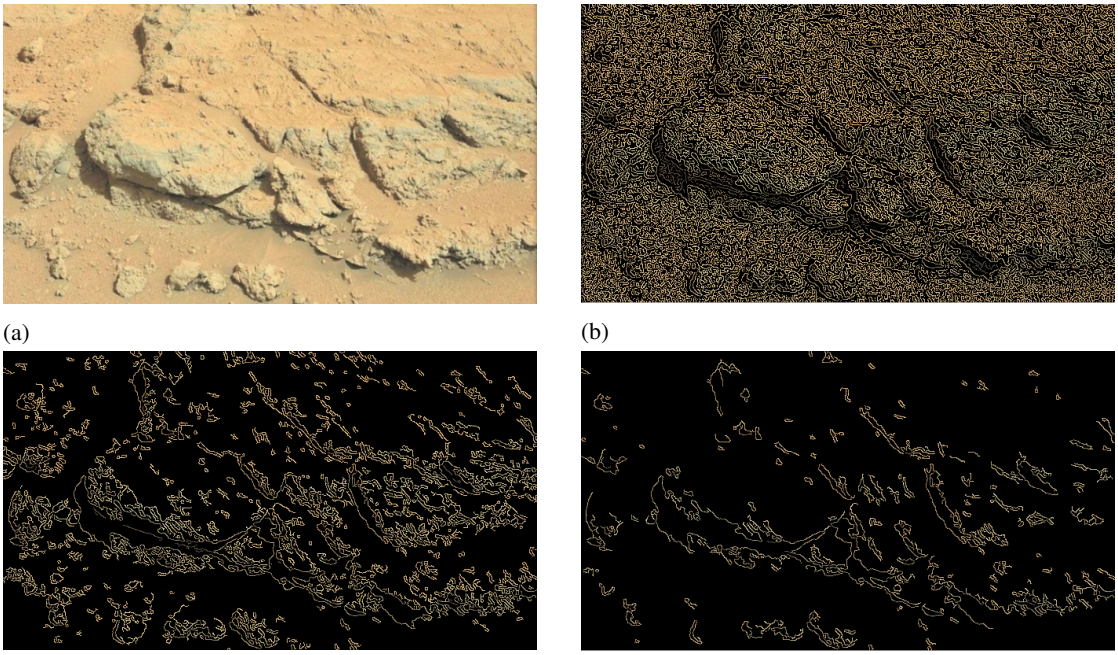
Since the ranges used by the object finder are quite narrow, the resulting binary image contains a number of small groups of connected white pixels. To connect groups that likely belong to the same surface and filter out insignificant ones, the object finder performs an erosion and dilation. It takes into account the size of all the remaining groups of color and draws a bounding box around the largest one. The center of that bounding box is then assumed to be the center of the object. The depth value at the center's coordinates in the depth image is taken to be the z-coordinate of the object.

The object finder estimates the position of all three objects. For each of the objects it registers the calculated center as the position of the object in the tf tree. During its initialization, the particle filter reads that position from the tf tree and initializes the particles in a small radius around it.

3.1.2. Number of particles

The choice of how many particles to use is quite difficult. The state space is very large and would require a large number of particles to be covered. However, every single particle has to be evaluated in every iteration, which is computationally very expensive. Since the particle filter uses the output from the object finder to delimit the search space, less particles are needed to cover the search space.

As Isard and Blake [12] already proposed and implemented in their work, the number of particles does not need to be fixed. They start with a high number and gradually reduce it. This particle filter has an upper and a lower bound on the number of particles. Initially, it starts at the upper bound. In each iteration, it deletes a fix number of the particles with the least beliefs, until the lower bound is reached. Every 100th iteration, the number of particles increases again to the original amount, i.e. the upper bound. The new particles are placed within a radius of the position estimate from the object finder, as if it was the first iteration. The number of particles is again slowly reduced until the number of particles reaches the lower bound. This reinitialization is supposed to help recovering from losing track of the object or being stuck in a local minimum. Fresh particles could get a better estimate with a higher belief than old particles and alter the posterior distribution in a significant way. When the number of particles increases, more of the state space can be covered, which potentially increases the chances of a good estimate. In chapter 4, I will show the performance of the particle filter using different upper limits of numbers of particles.



(a) (b) (c) (d)

Figure 3.1.: Result of applying the Canny edge filter to a typical image of a planetary surface.

3.2. Observation

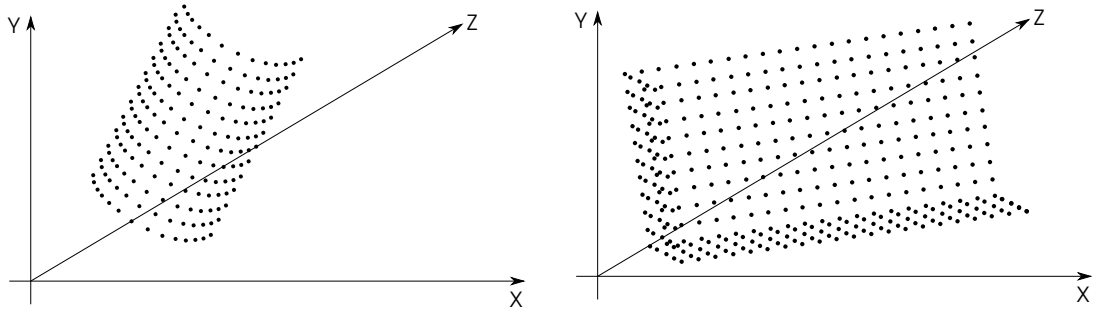
In the observation stage of the particle filter the likelihood of the particles is determined by evaluating the current observations. In this application the observations consist of RGB and depth images from the Kinect.

3.2.1. Object models

Using edge features is very common in object recognition and tracking applications ([13], [16], [19], [22]). This particle filter does not use edge features, as they have the following issues. This particle filter is meant to be used for object pose estimation on planetary surfaces. A setting like that will probably contain rocks of various sizes. Figure 3.1 shows the results of applying a Canny edge filter to a picture of such a setting. Figure 3.1a is the original.[1] Figure 3.1b is the result of a Canny filter applied to the original, with a very low minimum threshold. Figure 3.1c is the result with a moderate threshold and figure 3.1d is the result with a high threshold. It is visible that even with a high threshold there is a significant edge clutter, which "can lead to a significant rate of false alarms."[19, p. 103] More general arguments against edge features are their "lack of robustness to motion and optical blur"[27, p. 785] and the lack of good description techniques for edge features.[13, p. 1119]

Instead of modelling the objects using their edges, the object model of this thesis is rather similar to the object model of Taiana et al. [27]. They randomly pick a number of pixels from within and outside of the

[1] Image from [1].



(a) Model of the cup

(b) Model of the battery

Figure 3.2.: 3D object models.

projected object's surfaces and evaluate their color values. The object model in this thesis also consists of pixels from within and outside the projected object surfaces, although they are not randomly picked.

First, a grid consisting of uniformly distributed 3D points placed on the surface of the object is generated. This grid is located at the origin of the coordinate frame and not rotated. Since the camera can only see the front half of a 3D object, only points on the front facing surfaces are generated (illustrated in figure 3.2). Note that the grid is created before shifting and rotating so the modelled half may not actually be the front half (see figure 3.2b). That behavior is tolerated for the sake if simplicity and in the hopes that particles with unfavorable rotations are eliminated during resampling. In the next step, the grid is shifted to its designated location and rotated by its given angles. When the object model is positioned at its designated coordinates, it is projected into 2D space using a pinhole camera model.

3.2.2. Evaluation

The belief of a particle is calculated based on the current observation. The observation includes RGB as well as depth data. Han et al. [11] advocate strongly for the combination of depth and color data, arguing that their traits complement each other and naming several studies that benefited from combining these two. While depth information has an obvious advantage when trying to find the 3D position of an object, color information has the advantage of robustness against motion blur and optical blur (cf. Taiana et al. [27]) and robustness against partial occlusion (cf. Nummiaro et al. [18]).

After projecting the grid points of the object into 2D space with the pinhole camera model, each projected point is tested for two qualities; its depth value and its color value. Points that are projected outside of the 2D image are ignored.

Since this particle filter is meant to be used in a scenario in which the object is in close proximity to the robot and needs to be grasped, depth values that are too far away are irrelevant. If the depth value at the coordinates of the grid point in the depth image is too high (1 m), the particle receives a high penalty. The same penalty is given if the depth value is too low (5 cm). If it is neither too far nor too close, the difference between the image's depth value and the grid point's depth value is evaluated in two steps. The first step checks whether the difference is greater than 2 cm. If that is the case, there is a small penalty. A higher penalty is given, if the difference is greater than 3.5 cm. If the difference is even greater than that,

the grid point gets the maximum penalty.

$$d_g = \begin{cases} 1.5, & \text{if } D(x(g), y(g)) > 1 \text{ or } D(x(g), y(g)) < 0.05 \\ 1, & \text{if } ||z(g) - D(x(g), y(g))|| > 0.035 \\ 0.3, & \text{if } 0.035 > ||z(g) - D(x(g), y(g))|| > 0.02 \\ 0, & \text{otherwise} \end{cases}, \tag{3.2}$$

where $x(g)$ and $y(g)$ are the 2D coordinates of the projection of g and $z(g)$ is the depth value of the grid point g. D is the depth image and $D(x(g), y(g))$ is the depth value at the coordinates $x(g), y(g)$. This differentiation is meant to prevent the early discarding of candidates. The scores of all pixels within the 2D image are accumulated and divided by the number of image pixels, i.e. the points that are projected into the 2D image. The overall score for the depth information s_D is

$$s_D = \frac{1}{M} \sum_{m=1}^{M} d_{g_m}, \tag{3.3}$$

where M is the number of grid points that are projected into the 2D image and g_m is the m-th grid point.

Prior to the evaluation of the color data, the RGB image is converted into HSV color space. The `inRange(...)` function from OpenCV is used to segment the object's color from the converted image. Each grid point is then examined for the value at its corresponding 2D coordinates in the resulting binary image. The pixels with a matching color value are counted and the sum is divided by the number of grid points that are projected into the 2D image.

The overall score for the color information s_C is

$$s_C = \frac{1}{M} \sum_{m=1}^{M} C(x(g_m), y(g_m)), \tag{3.4}$$

where C is the binary image resulting from the color segmentation.

Apart from considering pixels within the object's projection, Taiana et al. [27] also evaluate pixels in the vicinity of the object. The vicinity can give additional information about the likelihood of the particle. In the case of the battery, any flat surface could be mistaken for the battery's surface. Since the pixels around the object don't belong to the object, they are expected to have different color and depth values than the object's pixels. Thus, if the pixels of the object's vicinity are too similar to the object's surface's pixels, that is an indicator for the particle being an unlikely candidate. Therefore, without considering the vicinity, there is a chance of a higher rate of false positives. That is why this particle filter evaluates pixels outside the object's projection as well.

While Taiana et al. pick the pixels randomly, this approach looks at a few specifically chosen pixels. As a first step, the grid points with the highest and lowest x and y coordinates are determined. This should yield the four grid points with the most remote coordinates. These four points are then projected into 2D space. Starting from the 2D coordinates of each corner, the algorithm walks 30 pixels vertically or horizontally outward, away from the object. Figure 3.3 illustrates which pixels are chosen. $g_{up}, g_{down}, g_{left}, g_{right}$ mark the most outward grid points. y_{min} is the y coordinate of the highest grid point, y_{max} is the y coordinate of the lowest grid point, x_{min} is the x coordinate of the left-most grid point, x_{max} is the x coordinate of the right-most grid point. Starting at each of these points, 30 pixels are chosen in the

upward direction from y_{min}, downward from y_{max}, left from x_{min} and right from x_{max}. The thick black bars show the position of these pixels. These pixels of course don't cover the complete surroundings, but they should give a sense of what the vicinity approximately looks like. A margin of 30 pixels should be enough to cover the relevant environment.

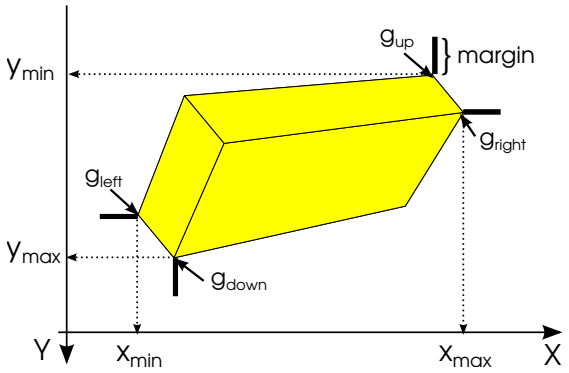


Figure 3.3.: Illustration of which pixels are considered for the evaluation of the object's vicinity.

These pixels are evaluated by the same principle as the ones within the object's projection. The number of pixels with the correct color (i.e. a color that does not match the color of the object) is divided by the number of all vicinity pixels within the 2D image.

The score $s_{V,C}$ of the color evaluation of the vicinity is

$$s_{V,C} = \frac{1}{V} \sum_{i=1}^{V} 1 - C(x(v_i), y(v_i)), \tag{3.5}$$

where v_i is the i-th pixel of the vicinity with the coordinates $x(v_i)$ and $y(v_i)$, and V is the number of chosen pixels from the vicinity that are within the limits of the 2D image.

The depth evaluation of the vicinity pixels depends on their location w.r.t. the object. A point below the object is to be expected to be closer to the camera and have a lower depth value than the object. Points above, left or right of the object would rather be expected to be further away from the camera than the object itself and therefore have a higher depth value.

For each direction, the average depth value of the 30 pixels is calculated and compared to the depth value of the corresponding grid point.

$$avg_{dir} = \frac{1}{V_{dir}} \sum_{j=down}^{V_{dir}} D(x(v_{dir,j}), y(v_{dir,j})), \tag{3.6}$$

where dir is an element of $\{up, down, left, right\}$ indicates the direction w.r.t. the object, V_{dir} is the number of pixels per direction dir whose coordinates are within the 2D image, and $x(v_{dir,j})$ and $y(v_{dir,j})$ are the coordinates of the j-th vicinity pixel $v_{dir,j}$ in direction dir. If the ratio of the two depth values is

as expected, the overall color score for the vicinity is increased by one.

$$s_{dir} = \begin{cases} 1, & \text{if } avg_{dir} > z(g_{dir}) \text{ and } dir \neq \text{ down} \\ 1, & \text{if } avg_{dir} < z(g_{dir}) \text{ and } dir = \text{ down} \\ 0, & \text{otherwise} \end{cases}, \tag{3.7}$$

where $z(g_{dir})$ is the depth value of the grid point that was the starting point for the respective direction. Upon adding up the score for all four directions, the result is normalized.

$$s_{V,D} = \frac{1}{4} \sum_{dir=1}^{4} s_{dir}. \tag{3.8}$$

The belief b combines the color and depth scores of the object's projection and vicinity:

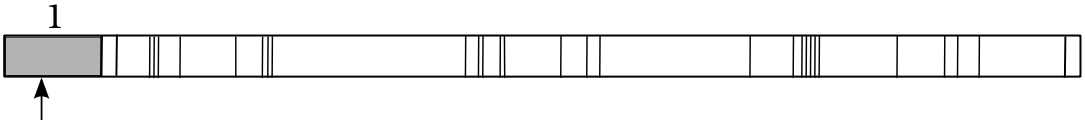
$$b = exp\left(-\frac{s_D}{0.05}\right) \cdot exp\left(-\frac{s_C}{0.1}\right) \cdot s_{V,D} \cdot s_{V,C}. \tag{3.9}$$

s_C and s_D are modified one last time to have a lower tolerance for the depth and color data of the object itself. The task is to detect the pose of the object as accurately as possible. Detailed information about shape and color of the objects is given, but the environment in which it is positioned is only vaguely known beforehand. It is therefore only possible to look for objects that are as similar to the models as possible and to check that the surrounding area does not include objects that match the models too well, in order to prevent false positives. Since the vicinity can only be judged for the absence of characteristics without having detailed knowledge about what characteristics should be there, there are less criteria that the vicinity can be judged by. Therefore it is not useful to apply very strict criteria to the data from the vicinity. The algorithm is more tolerant with the color score, as the color data is more susceptible to environmental changes such as changes in lighting than depth data.

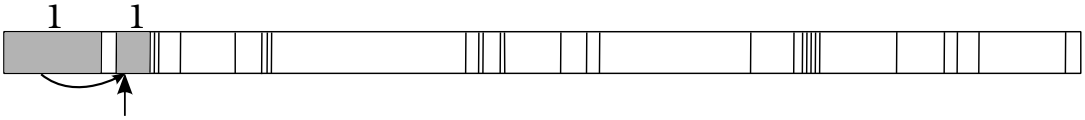
3.3. Resampling and prediction

The beliefs that have been computed in the observation stage are used for the computation of the importance weights in the resampling step. The purpose of the resampling stage is to promote promising particles and eliminate irrelevant ones. This resampling algorithm is a SIR algorithm and closely related to the approaches of Gordon et al. [10] and Isard and Blake [12]. Imagine all particles lying in a row, each one as wide as its belief, as illustrated in figure 3.4. The particle filter first computes the average belief and the starting point is assigned a random value between zero and the average belief (see figure 3.4a). The particle that the starting point points to, has its weight increased by one (indicated by the number above the field). The starting point is shifted by the average belief (see figure 3.4b). This process is repeated until the end of the row is reached (see figure 3.4c). Each particle that it stops at, also has its weight increased by one. Particles with high beliefs may be picked multiple times, but particles with very low beliefs also have a chance to be picked. The distribution of the weights will be roughly proportional to the distribution of the beliefs. Due to the random starting point, SIR is a Monte Carlo method.

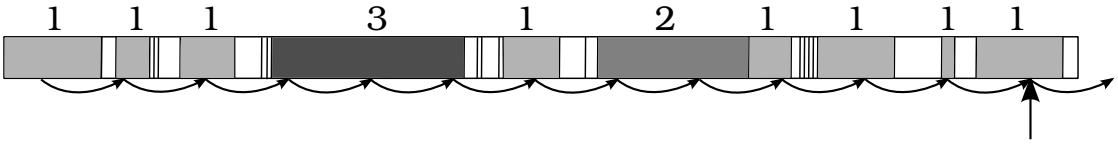
Now that the weights are assigned, the low-weighted particles can be eliminated. Particles with the weight 1 are shifted according to the motion model, particles with even higher weights are copied as many times



(a) The starting point is randomly chosen between zero and the average belief.



(b) The arrow is shifted by the amount of the average belief.



(c) The arrow is shifted by the same amount as long as there are particles left.

Figure 3.4.: Process of assigning importance weights to particles.

as their weight dictates. These copies are also shifted according to the motion model. The copies of the high-weighted particles take the places of particles with weight zero. If there still are particles left that have weight zero, they are newly initialized, i.e. moved to a random pose in the state space. As a final step, the 10% of the particles with the lowest beliefs are newly initialized. This step is supposed to help in case the particle filter is stuck in a local minimum, because a randomly initialized particle may find a better estimate. The pseudo code of the complete process can be found in section A.1.

The motion model predicts the movement of the object in the state space. For example, if a robot has a maximum speed of 3 km/h, the likelihood of it being more than 3 km away from the starting point after one hour is very low. The prediction can focus on the area within a 3 km radius and no particles have to be created further away. The exact movement of this project's robot is unknown at the current stage, but it is known that the robot moves rather slowly. The motion model of this particle filter therefore shifts the particles by a random value within a small radius of 10 cm and a rotates them by a random angle between 0° and 30° around each axis.

3.4. Implementation details

This particle filter was implemented in C++. The Eigen library [6] was used for rotation and translation of the object models, and the OpenCV library [20] was used for image processing tasks like color segmentation.

This implementation was developed with the ROS version *groovy*. For the Kinect, the *freenect* driver from the ROS framework was employed [25]. The driver provides the topics that the particle filter subscribes to: a topic containing depth data[2], a topic containing RGB data[3], and a topic containing information about the camera model used by the pinhole camera model[4], which were used by the particle filter. The image size from these topics is 640 x 480 pixel, the raw image data of is 1280 x 480 byte.

[2] `/camera/depth_registered/image_rect_raw`
[3] `/camera/rgb/image_color`
[4] `/camera/depth_registered/camera_info`

For random number generation, the `mt_19937` engine from the `<random>` header from the C++ standard library is used. The seed for the engine is a `random_device`, and the distribution is a `uniform_real_distribution`. The `mt_19937` engine is an implementation of the Mersenne Twister, a pseudorandom number generator with a period length of $2^{19937} - 1$. Its advantages are its speed and the equidistribution of its output. Its disadvantages are that it works with a large state space and therefore occupies a lot of cache, and also that it is not cryptographically secure, although the latter characteristic should be negligible in this particular application.

4. Experiments

This chapter examines the performance of the implemented particle filter. It first describes the way in which the experiments were conducted. Subsequently the results of these experiments are presented, followed by a discussion thereof.

As a beginning, the requirements for the particle filter need to be specified. This particle filter was implemented to estimate the pose of an object from an online image stream. Thus, it has to be fast enough to process real-time data. The object's pose should be estimated with a sufficient accuracy so that the robotic arms and their grippers are able to reliably grasp the object while using the estimated pose. Especially the gripper for the cup requires a high level of accuracy, because the sling has to be positioned directly above the cup. An acceptable error would be 0 cm to 2 cm in position and 0° to 10° in orientation. Furthermore, that level of accuracy has to be maintained despite motion blur, i.e. a moving camera and despite a moderate occlusion of the object.

4.1. Test set-up and execution

The test environment simulates the conditions of the SpaceBot Cup. It consists of a landscape with a few elevations. The landscape was covered with cloth that is approximately the color of sand or rocks. A picture of the test environment can be seen in figure 4.1.

Figure 4.1.: Test scene

The objects are positioned at a distance from the camera between 40 cm, which is the closest distance at which the Kinect can see depth values, and 100 cm which is the furthest distance that is deemed relevant. Both object types are placed in the scene, to ensure the same lighting conditions for both of types. To guarantee equal conditions for each test run, the ROS tool rosbag was used to record sequences of the

topics from the Kinect. The sequences contained all topics necessary for the particle filter and the object finder and were later played back as input for the two nodes.

Two different kinds of tests were conducted; tests for a rigid camera and a moving camera. This distinction was deemed important since the available data in the two cases necessitates a different kind of evaluation of the particle filter's performance.

4.1.1. Rigid camera

In this type of test, the object does not change position w.r.t the camera. Therefore, it's true position and orientation are known throughout the whole sequence. This circumstance permits a direct comparison with the estimated pose by the particle filter. The estimation of the particle filter, i.e. the weighted average of all particles per iteration is written into a file. Each test run has 400 iterations, so that one file contains the estimation for all 400 iterations for one test run.[1]

72 different test cases were conducted to test the particle filter's performance under different circumstances. Each test case was run 50 times. The test cases arose from the combination of all factors that were to be tested. The objects were placed at 3 different places in the scene, so that a potential bias for certain poses would be less influential (for details, see appendix B.2). Each pose was recorded once without occlusion and once with occlusion of the objects. The particle filter was run once with consideration of the vicinity and once without. Also, different upper bounds for the number of particles were tested; 100 (no variation in number of particles, since the lower bound was always 100), 300 and 500. A complete list of all test cases can be found in the appendix B.1.

The error between actual pose and estimated pose will be divided into an error for position and an error for orientation because the two have inherently different scales (0 m - 1 m for position, 0° - 90° for orientation[2]). The difference between two positions was simply the euclidean distance between them. The difference between two orientations was calculated with the following equation:

$$e = arccos\left(\frac{o1 \cdot o2}{\|o1\|\|o2\|}\right)\frac{180}{\pi}, \tag{4.1}$$

where $o1$ and $o2$ are the two vectors describing the orientation of two different particles.

4.1.2. Moving camera

When the object does change its position w.r.t to the camera, it is not possible to obtain data about the real pose of the objects throughout the complete image stream. Only the visual output can be used to evaluate the performance of the particle filter with a moving camera. A sequence of the test environment was recorded with rosbag and played back to the particle filter and the object finder. The particle filter computes the weighted average of all particles and creates a new particle with the coordinates of the weighted average. A grid for this particle is generated and the individual grid points are projected into 2D space. The respective pixels in the depth image are marked.

[1] The recorded data sequences are between 60 and 70 seconds long. In case the sequence ended before the completion of 400 iterations, it was simply played once more. That is only possible because the objects in the image stream don't move.

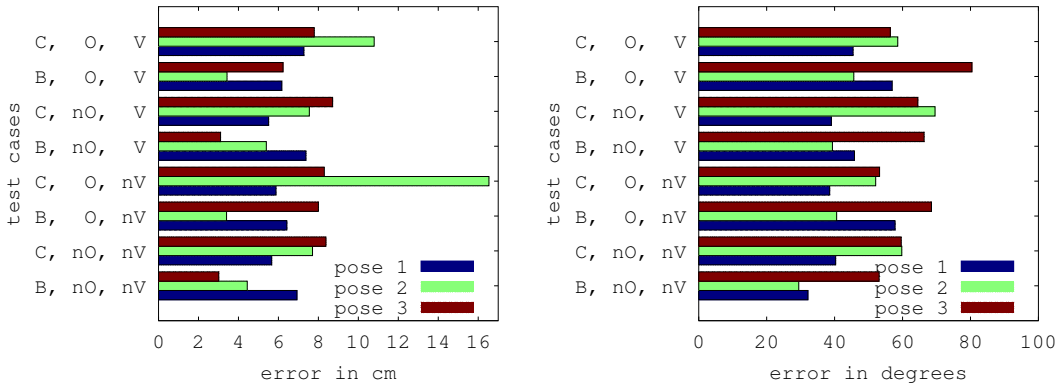
[2] The maximum error in orientation is 90° due to the symmetry of the objects.

4.2. Results

This section presents the results of the experiments regarding different aspects. All 72 test cases are executed with a lower bound for the number of particles of 100, only the upper bound is varying. Each test run has 400 iterations, to observe the behavior of the particle filter over time. The bar plots show the error averaged over all iterations. Figures which show the error development over time can be found in appendix C.

4.2.1. Object poses

In the experiments, the objects were positioned at three different places, with different orientations. This was done to see whether the particle filter had a bias towards certain poses. As observable from figure 4.2[3], the results show no clear tendency to favor one or the other pose. Thus, there seems to be no definite bias for one of the poses. Therefore, in the sections 4.2.2, 4.2.3, 4.2.4 and 4.2.5, the results are averaged over the three poses.



(a) Error in position

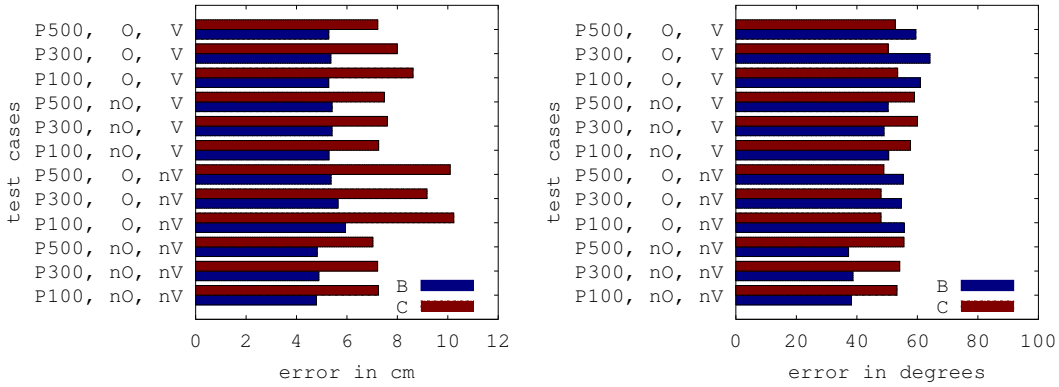
(b) Error in orientation

Figure 4.2.: Error for different poses. B = battery, C = cup, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

4.2.2. Object types

In almost all test cases, the battery achieves a lower error than the cup (see figure 4.3a). The error difference in position is approximately 2 cm on average. The cup only has better results when it is occluded, regarding the orientation error (see figure 4.3b).

[3]This figure and the figures in appendix C.1 only show results for the test cases with an upper bound of 100 for the number of particles. These test cases are sufficient to underline the argument.



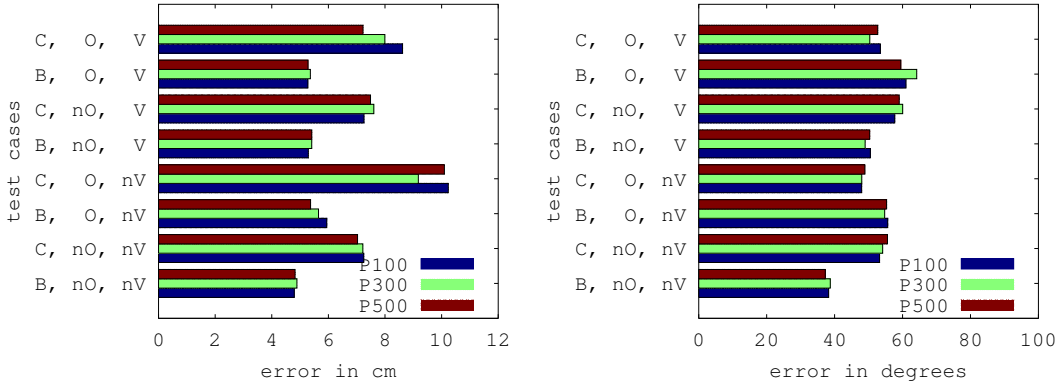
(a) Error in position

(b) Error in orientation

Figure 4.3.: Error for different object types. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

4.2.3. Number of particles

For all test cases, there is little to no difference in the performance of the particle filter regarding the different upper bounds for the number of particles. Only the two test cases with the occluded cup show a noteworthy difference in performance, regarding position; with consideration of the vicinity between the upper bounds 100 and 300 and between the upper bounds 300 and 500, without consideration of the vicinity between the upper bounds 300 and 500 (see figure 4.4a).



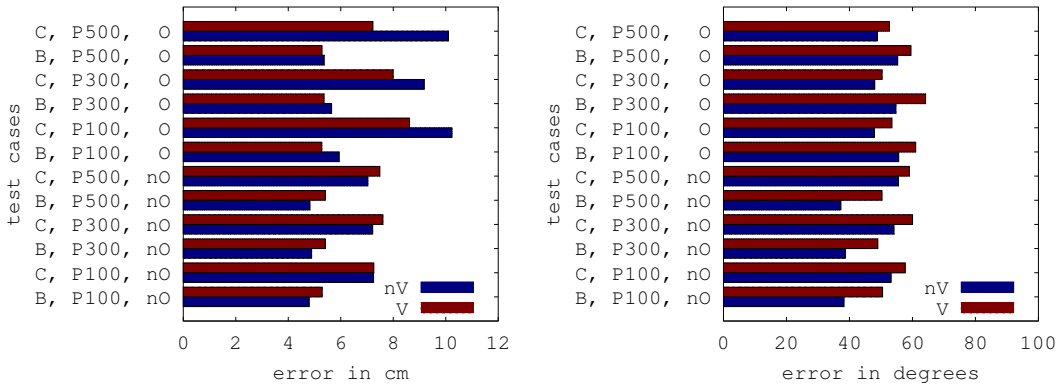
(a) Error in position

(b) Error in orientation

Figure 4.4.: Error for different upper bounds for the number of particles. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

4.2.4. Vicinity

While most test cases show no significant difference in performance of the particle filter regarding the consideration of the object's vicinity, there seems to be a tendency for better performance without the consideration of the vicinity. Only when the objects are occluded, considering the vicinity results in a noticeably lower error in position than when not considering it (see figure 4.5a).

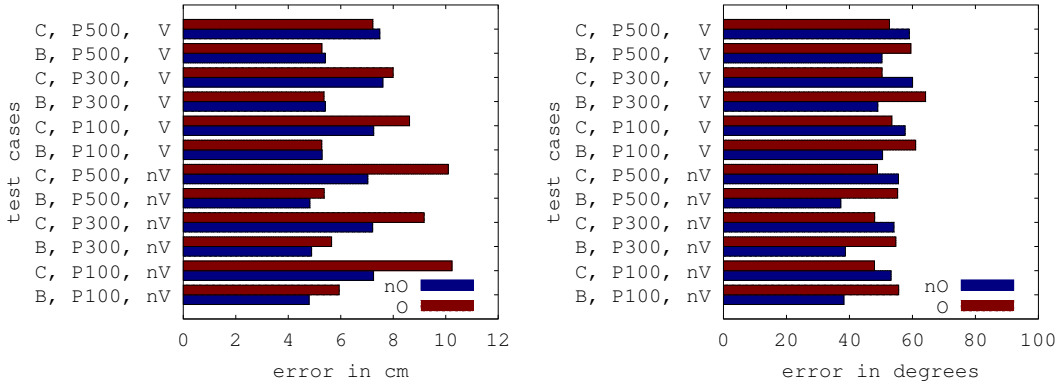


(a) Error in position

(b) Error in orientation

Figure 4.5.: Error for consideration/ no consideration of the vicinity. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

4.2.5. Occlusion



(a) Error in position

(b) Error in orientation

Figure 4.6.: Error for occluded objects/non-occluded objects. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

Almost all test cases show better results in position for non-occluded objects. Regarding orientation, the non-occluded battery results in a lower error than the occluded battery, but the occluded cup has a lower error than the non-occluded cup (see figure 4.6b).

4.2.6. Moving camera

As mentioned in section 4.1.2, the particle filter's performance with a moving camera can only be judged by its visual output. Therefore, a screencast was taken while the particle filter was applied to a previously recorded data sequence from the Kinect. To obtain an objective sample of screenshots, the screen was captured every 10 seconds from the moment the particle filter started. The particle filter was run with an upper bound of 500 particles and with the consideration of the object's vicinity. The objects in the sequence are not occluded. Figure 4.7 shows the screenshots from the screencast. The left image shows the RGB image and the right image is a detail from the depth image that shows the output from the particle filter. The white pixels mark the grid points of the object that is the weighted average of all particles.

As visible in figure 4.7, the estimation of the particle filter for the orientation is not stable and often changes. The estimated position does not show such rapid changes. While it is difficult to derive the position error from the images, it is clear that the orientation error is quite high. In most images, the result is very inaccurate.

4.2.7. Run time

Resampling, updating the weights and the propagation of the particles are all linear with the number of particles. The evaluation of a particle is dependent on the number of grid points of a particle. With N being the number of particles and G being the number of grid points per particle, the complexity of this particle filter implementation is $\mathcal{O}(N \cdot G)$.

The runtime of the particle filter depends on the number of particles, on the number of grid points in the object model and on whether the vicinity is considered or not. Figure 4.8 shows the average duration of the different parts of an iteration of the particle filter, depending on these factors.

The creation phase is the phase in which new particles are created. It only occurs every 100th iteration. The evaluation phase includes the preprocessing of the RGB image, the evaluation of each particle and the normalization of all beliefs. In the resampling phase, the particle filter computes the new weights of the particles and propagates them according to the motion model.

The duration of the creation phase is negligible. Object type and consideration of the object's vicinity do not reflect in the duration of the resampling phase. The length of the evaluation phase is highly subjective to all 3 factors. When considering the vicinity, the particle filter evaluates more image pixels, resulting in a longer duration (see figure 4.8a). Regarding the object types, the particle filter takes longer for the battery. That is due to the fact that the grid points are uniformly distributed in a distance of 1 cm of each other and that the battery is the bigger object. More grid points also result in a longer runtime (see figure 4.8b). The duration of the evaluation phase increases with the number of particles.

(a) 0:03, iteration 2

(b) 0:13, iteration 30

(c) 0:23, iteration 55

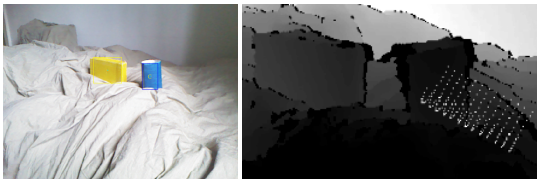
(d) 0:33, iteration 100

(e) 0:43, iteration 131

(f) 0:53, iteration 163

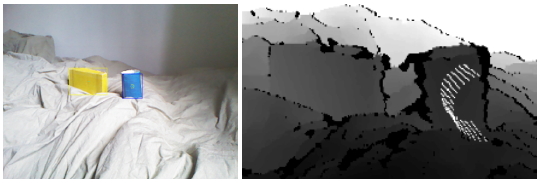
(g) 1:03, iteration 195

(h) 1:13, iteration 226



(i) 1:23, iteration 276

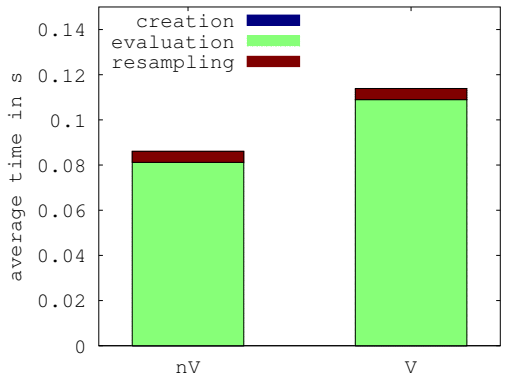
(j) 1:33, iteration 300



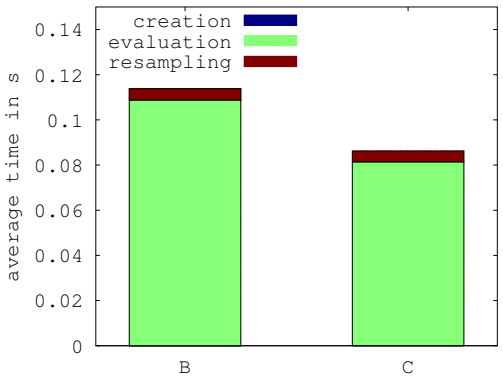
(k) 1:43, iteration 321

(l) 1:53, iteration 374

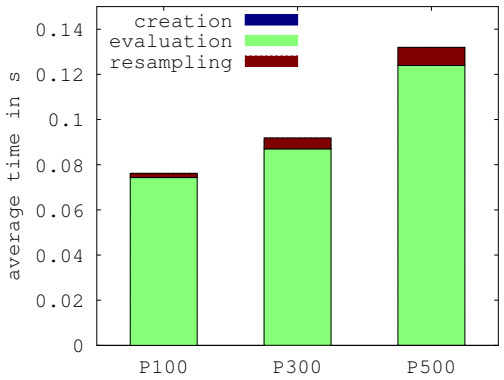
Figure 4.7.: Screenshots from the experiments for the moving objects



(a) Average duration for vicinity consideration



(b) Average duration for different object type



(c) Average duration for different numbers of particles

Figure 4.8.: Average duration of creation, evaluation and resampling of the particles, per iteration. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nV = no vicinity, V = vicinity

4.3. Discussion

The test results are altogether unsatisfactory. The error of the estimation much higher than aspired. Neither position nor orientation are estimated with sufficient accuracy for a robot to grasp the object. Since the errors are so high in general, it is questionable whether the results can give meaningful insight into the influence of the different factors on the performance of the particle filter. If conclusions can be drawn, they are as follows.

The particle filter does not estimate both object types with the same accuracy. When the objects are not occluded, the particle filter performs considerably better for the battery than for the cup. The position error is always lower for the battery. However, when the objects are not completely visible, the cup has a lower orientation error than the battery.

The number of particles does not have as much influence as expected. The performance does not improve significantly with a higher upper bound for the number of particles. Instead, it is almost the same. At the iterations where the new particles are created, a drop in the error would have been expected. However, such a drop is only barely visible in some of the figures. Considering the increase in processing time when choosing a higher upper bound, using more particles does not seem profitable.

The consideration of the object's vicinity also does not result in the anticipated behavior, which is increased performance when considering the vicinity. Surprisingly, the performance only improves when the objects are occluded, and the improvement only reflects in the position error.

As for occlusion, for the most part the particle filter behaves as expected, i.e. if performs better with non-occluded objects. However, regarding orientation, the error for the cup is lower when it is occluded compared to when it is completely visible.

Considering the results for the experiments with a rigid camera, it is not surprising that the results for a moving camera are inadequate. The estimation is not only unstable and often changes, but is also inaccurate.

Regarding the runtime of the particle filter, the average duration of one iteration can not be called fast. Nonetheless, it should be acceptable in the specific application scenario, as the robot is moving neither a lot nor fast. Thus, there are no rapid changes to be expected in the image stream, and it is unlikely for the robot to miss a significant amount of information in the case that a frame cannot be processed.

There are several possible, complementary explanations for the test results. The first explanation is that during the conduction of the experiments, the measuring of the true object poses may have been faulty. Secondly, the object models are not sophisticated. To keep the models simple, only half of the object's surface was modelled and then rotated, in the hopes that the resampling process would filter out the implausible candidates. That approach has not proven successful. A third explanation could lie in the calculation of the belief, as the formula did not reliably recognize good candidates. Another reason could be the motion model, since it was very rudimentary.

5. Recapitulation

This thesis presents the approach that was taken for the estimation of an object pose. The author of this thesis implemented a particle filter that uses both color and depth information provided by a Kinect for Windows™. The objects were modelled as a set of 3D points, that are placed on the visible surface of the respective object, i.e. visible from the point of view of a camera. Additionally to these grid points, some pixels from the vicinity are evaluated to reduce the rate of false positives.

Judging by the experiment results in chapter 4, that approach was not successful. The error in both position and orientation are higher than tolerable in the application scenario. Especially the design of the gripper of the robotic arm for the cup does not allow much error. The results are probably caused by a number of different factors. Concerning the object models, the grid points were placed before rotating the grid. As a consequence, particles with a rotation such that the grid points ended up on the back of the object were possible. That risk was taken in the hopes that such particles would be eliminated during the resampling stage. It seems that this strategy was not successful. Another factor may have been the calculation of the belief. The weighing of the various scores may have been too strict. Consequentially, good candidates may not have been given a belief high enough so that they would have stood out during the resampling stage. A third factor may have been the rudimentary nature of the motion model. A mismatch between the motion model and the actual motion can lead to significant performance differences. Since the number of particles is known to be influential to a particle filter's performance, it is likely, that the amount of particles used was too low. It was attempted to circumvent using too few particles by regularly initializing new particles. Yet, with the run time in mind, the overall number of particles is also reduced again to a lower bound. Possibly, that lower bound was too low.

Disregarding the high level of the error, a way to improve the particle filter's performance may be to adjust its algorithm to the situation. Since the type of object is known beforehand, it would be possible to configure the particle filter. For example, the consideration of the vicinity only increased the advantage for the cup, but not for the battery. Therefore, the algorithm could only consider the vicinity if it is looking to estimate the pose of the cup. Furthermore, it would be interesting to see, if the performance is improved if the lower bound of the number of particles is increased.

However, considering the poor performance of the particle filter, it is obvious that several aspects of the concept should be redesigned or refined.

Bibliography

[1] CENTRALES ERFORSCHUNGS-NETZ AUSSERGEWÖHNLICHER HIMMELS-PHÄNOMENE. Mars-Curiosity-Chroniken - Curiosity-News Sol 389-400. http://www.hjkc.de/_blog/2013/10/16/mars-curiosity-chroniken---curiosity-news-sol-389-400/, Oct. 16 2013. accessed on July 4, 2014.

[2] CHEN, Z. Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics 182*, 1 (2003), 1–69.

[3] COLLET, A., MARTINEZ, M., AND SRINIVASA, S. S. The MOPED framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research 30*, 10 (2011), 1284–1306.

[4] DEUTSCHES ZENTRUM FÜR LUFT- UND RAUMFAHRT. SpaceBot Cup 2013. Robotik-Wettbewerb des Raumfahrtmanagements im DLR. Wettbewerbsszenario und Aufgabenbeschreibung. Regularien und Bewertung. Unpublished draft of regulations for the SpaceBot Cup, handed out to the teams, 2013.

[5] DOUCET, A., AND JOHANSEN, A. M. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering 12* (2009), 656–704.

[6] EIGEN. http://eigen.tuxfamily.org/. accessed on June 22, 2014.

[7] EKVALL, S., HOFFMANN, F., AND KRAGIC, D. Object recognition and pose estimation for robotic manipulation using color cooccurrence histograms. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on* (2003), vol. 2, IEEE, pp. 1284–1289.

[8] FRAUNHOFER-INSTITUT FÜR PRODUKTIONSTECHNIK UND AUTOMATISIERUNG IPA. Press release Automatica 2014. ROS: software components for industrial robotics. http://www.ipa.fraunhofer.de/ROS_Industrial_Software_fuer_die_Industrierobotik.2750.0.html?&L=2, 2014. accessed on 5.7.2014.

[9] FREEDMAN, B., SHPUNT, A., AND ARIELI, Y. Distance-varying illumination and imaging techniques for depth mapping, Nov. 18 2010. US Patent App. 12/522,176.

[10] GORDON, N. J., SALMOND, D. J., AND SMITH, A. F. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)* (1993), vol. 140, IET, pp. 107–113.

[11] HAN, J. Enhanced computer vision with Microsoft Kinect sensor: A review. *IEEE Transactions on Cybernetics 43*, 5 (oct 2013), 1318–1334.

[12] ISARD, M., AND BLAKE, A. Condensation — conditional density propagation for visual tracking. *International Journal of Computer Vision 29*, 1 (1998), 5–28.

[13] KLEIN, G., AND MURRAY, D. W. Full-3D edge tracking with a particle filter. In *BMVC* (2006), pp. 1119–1128.

[14] MICROSOFT. PrimeSense supplies 3-D-sensing technology to "Project Natal" for Xbox 360. http://www.microsoft.com/en-us/news/press/2010/mar10/03-31primesensepr.aspx, Mar. 31 2010. accessed on June 22, 2014.

[15] MICROSOFT DEVELOPER NETWORK. Coordinate Spaces. http://msdn.microsoft.com/en-us/library/hh973078.aspx. accessed on July 4, 2014.

[16] MÖRWALD, T., ZILLICH, M., AND VINCZE, M. Edge tracking of textured objects with a recursive particle filter. In *Proceedings of the Graphicon* (2009).

[17] MURPHY, K. P. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

[18] NUMMIARO, K., KOLLER-MEIER, E., AND VAN GOOL, L. An adaptive color-based particle filter. *Image and vision computing 21*, 1 (2003), 99–110.

[19] OLSON, C. F., AND HUTTENLOCHER, D. P. Automatic target recognition by matching oriented edge pixels. *Image Processing, IEEE Transactions on 6*, 1 (1997), 103–113.

[20] OPENCV. http://opencv.org/. accessed on June 22, 2014.

[21] PCL. http://pointclouds.org/. accessed on June 25, 2014.

[22] PUPILLI, M., AND CALWAY, A. Real-time camera tracking using known 3d models and a particle filter. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on* (2006), vol. 1, IEEE, pp. 199–203.

[23] REINBACHER, C., RUTHER, M., AND BISCHOF, H. Pose estimation of known objects by efficient silhouette matching. In *Pattern Recognition (ICPR), 2010 20th International Conference on* (2010), IEEE, pp. 1080–1083.

[24] ROBOT OPERATING SYSTEM. http://www.ros.org/browse/list.php. accessed on June 25, 2014.

[25] ROBOT OPERATING SYSTEM. http://wiki.ros.org/freenect_stack. accessed on August 7, 2014.

[26] ROBOT OPERATING SYSTEM. tf. http://wiki.ros.org/tf. accessed on May 5, 2014.

[27] TAIANA, M., SANTOS, J., GASPAR, J., NASCIMENTO, J., BERNARDINO, A., AND LIMA, P. Tracking objects with generic calibrated sensors: an algorithm based on color and 3D shape features. *Robotics and autonomous systems 58*, 6 (2010), 784–795.

[28] VELODYNE LIDAR, I. *High Definition LiDAR™ HDL-32E*. http://velodynelidar.com/lidar/hdlproducts/hdl32e.aspx, 2013. accessed on July 4, 2014.

A. Algorithms

A.1. Resampling

The following pseudo code is describes the resampling algorithm of this particle filter.

Algorithm 1 Resampling

delete the 10 particles with the lowest beliefs
sumBeliefs ← sum of all beliefs
stepSize ← average of all beliefs
startPoint ← random value between 0 and stepSize
accumulatedBelief ← 0
numOfAssignedWeights ← 0
for $i = 0$ to N **do**
    accumulatedBelief ← accumulatedBelief + belief of particle p_i
    while accumulatedBelief $>$ start point **and** numOfAssignedWeights $<$ N **do**
        increase numOfAssignedWeights by 1
        increase weight of p_i by 1
        shift start point by step size
    end while
end for
for $i = 0$ to N **do**
    if weight of $p_i = 1$ **then**
        shift p_i in state space
    end if
    while weight of $p_i > 1$ **do**
        find first particle p_j with weight 0
        overwrite p_j with p_i
        shift p_j in state space
        decrease weight of p_i by 1
    end while
end for
for $i = 0$ to N **do**
    if weight of $p_i = 0$ **then**
        reinitialize p_i at random point in state space
    end if
end for
reinitialize the 10% particles with the worst beliefs

B. Experiment details

B.1. Test cases

The following table lists all test cases that were conducted in the experiments. See appendix B.2 for details about the different object poses.

Table B.1.: Test cases

#	object type	pose	max. # of particles	occlusion	vicinity
1	battery	1	100	y	y
2	battery	1	100	y	n
3	battery	1	300	y	y
4	battery	1	300	y	n
5	battery	1	500	y	y
6	battery	1	500	y	n
7	battery	1	100	n	y
8	battery	1	100	n	n
9	battery	1	300	n	y
10	battery	1	300	n	n
11	battery	1	500	n	y
12	battery	1	500	n	n
13	battery	2	100	y	y
14	battery	2	100	y	n
15	battery	2	300	y	y
16	battery	2	300	y	n
17	battery	2	500	y	y
18	battery	2	500	y	n
19	battery	2	100	n	y
20	battery	2	100	n	n
21	battery	2	300	n	y
22	battery	2	300	n	n
23	battery	2	500	n	y
24	battery	2	500	n	n
25	battery	3	100	y	y

Table B.1 – *Continued from previous page*

#	object type	pose	max. # of particles	occlusion	vicinity
26	battery	3	100	y	n
27	battery	3	300	y	y
28	battery	3	300	y	n
29	battery	3	500	y	y
30	battery	3	500	y	n
31	battery	3	100	n	y
32	battery	3	100	n	n
33	battery	3	300	n	y
34	battery	3	300	n	n
35	battery	3	500	n	y
36	battery	3	500	n	n
37	cup	1	100	y	y
38	cup	1	100	y	n
39	cup	1	300	y	y
40	cup	1	300	y	n
41	cup	1	500	y	y
42	cup	1	500	y	n
43	cup	1	100	n	y
44	cup	1	100	n	n
45	cup	1	300	n	y
46	cup	1	300	n	n
47	cup	1	500	n	y
48	cup	1	500	n	n
49	cup	2	100	y	y
50	cup	2	100	y	n
51	cup	2	300	y	y
52	cup	2	300	y	n
53	cup	2	500	y	y
54	cup	2	500	y	n
55	cup	2	100	n	y
56	cup	2	100	n	n
57	cup	2	300	n	y
58	cup	2	300	n	n
59	cup	2	500	n	y
60	cup	2	500	n	n
61	cup	3	100	y	y
62	cup	3	100	y	n

Table B.1 – *Continued from previous page*

#	object type	pose	max. # of particles	occlusion	vicinity
63	cup	3	300	y	y
64	cup	3	300	y	n
65	cup	3	500	y	y
66	cup	3	500	y	n
67	cup	3	100	n	y
68	cup	3	100	n	n
69	cup	3	300	n	y
70	cup	3	300	n	n
71	cup	3	500	n	y
72	cup	3	500	n	n

B.2. Object poses

(a) Pose 1

(b) Pose 2

(c) Pose 3

Figure B.1.: Object poses

Table B.2.: Poses of the battery

pose	x	y	z	roll	pitch	yaw
1	0.14	0.11	0.7	90	24	55
2	0.17	0.175	0.66	7	19	45
3	-0.03	0.23	0.75	6	91	45

x, y and z are given in meter. Roll, pitch and yaw are given in degrees.

Table B.3.: Poses of the cup

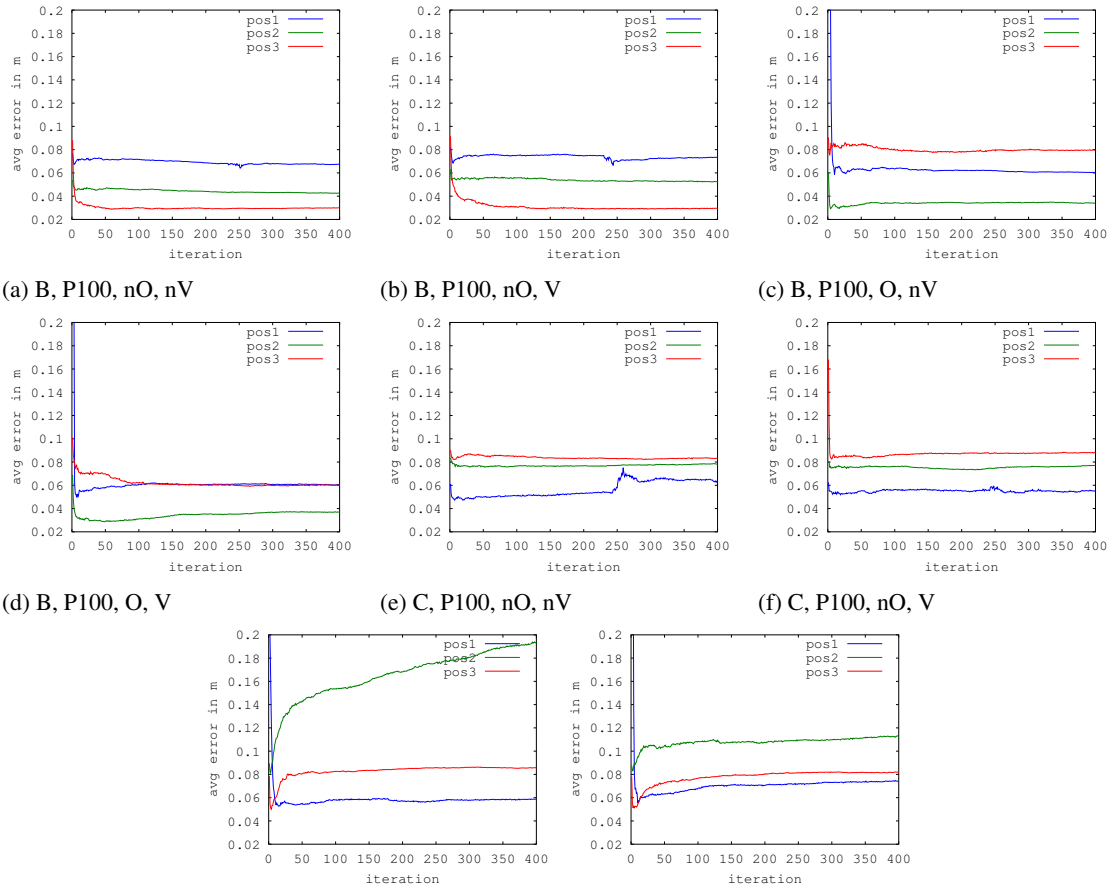
pose	x	y	z	roll	pitch	yaw
1	-0.08	0.22	0.775	0	100	55
2	-0.045	0.23	0.75	5	6	0
3	0.2	0.17	0.66	39	18	0

x, y and z are given in meter. Roll, pitch and yaw are given in degrees.

C. Experiment results

C.1. Object poses

This section only shows the results for the test cases with an upper bound of 100 for the particles.

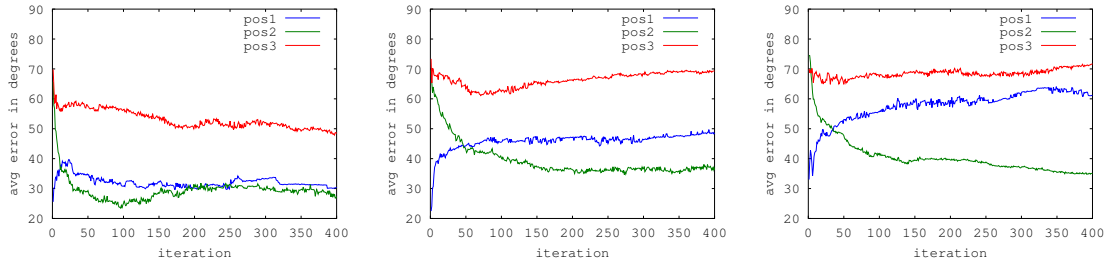


(a) B, P100, nO, nV (b) B, P100, nO, V (c) B, P100, O, nV

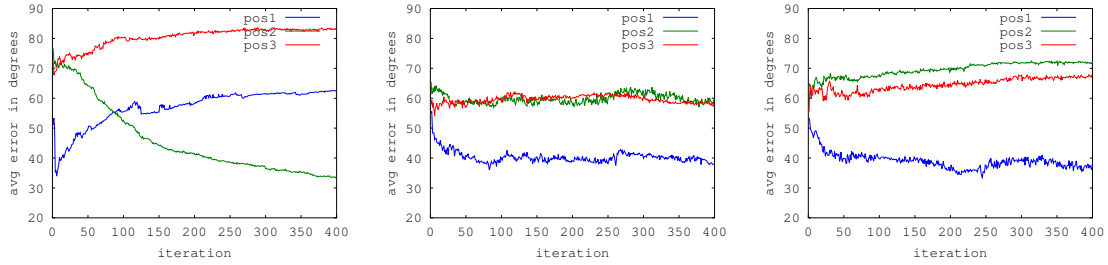
(d) B, P100, O, V (e) C, P100, nO, nV (f) C, P100, nO, V

(g) C, P100, O, nV (h) C, P100, O, V

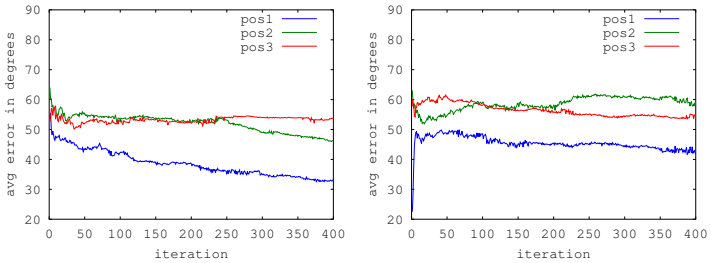
Figure C.1.: Error in position for different poses. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity



(a) B, P100, nO, nV (b) B, P100, nO, V (c) B, P100, O, nV



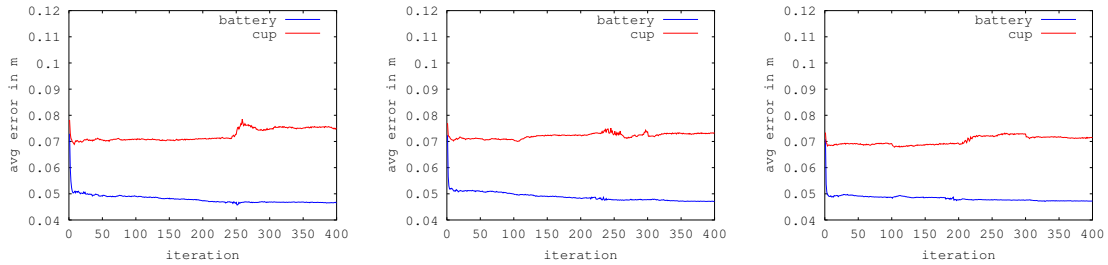
(d) B, P100, O, V (e) C, P100, nO, nV (f) C, P100, nO, V



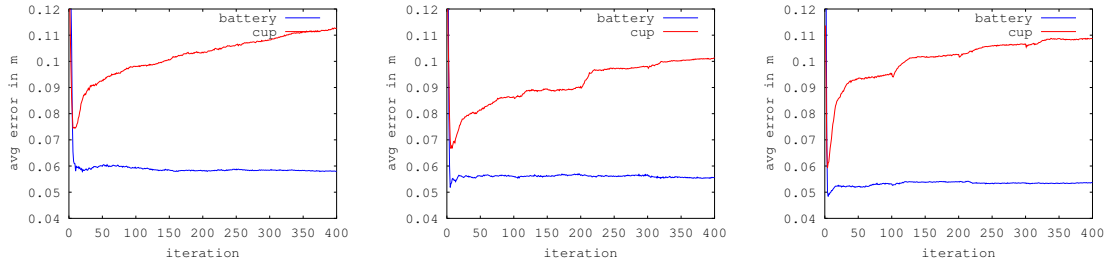
(g) C, P100, O, nV (h) C, P100, O, V

Figure C.2.: Error in orientation for different object poses. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

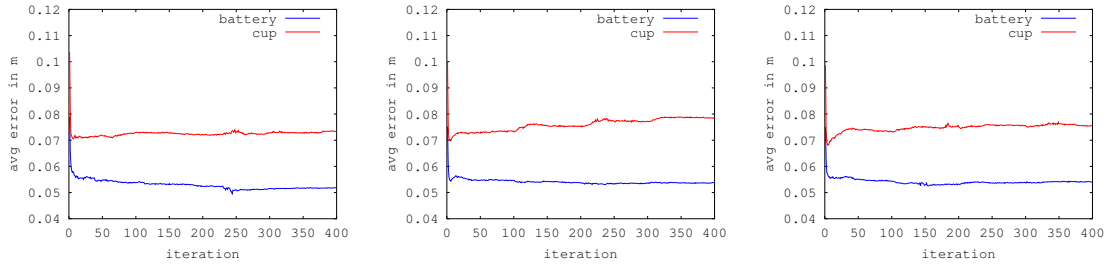
C.2. Object types



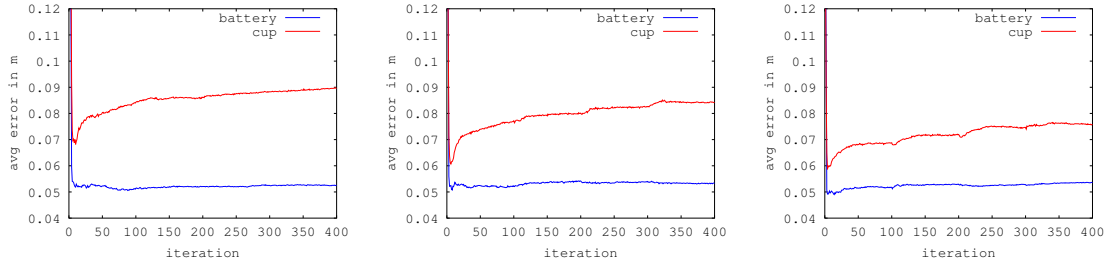
(a) P100, nO, nV (b) P300, nO, nV (c) 500 part., nO, nV



(d) P100, O, nV (e) P300, O, nV (f) P500, O, nV

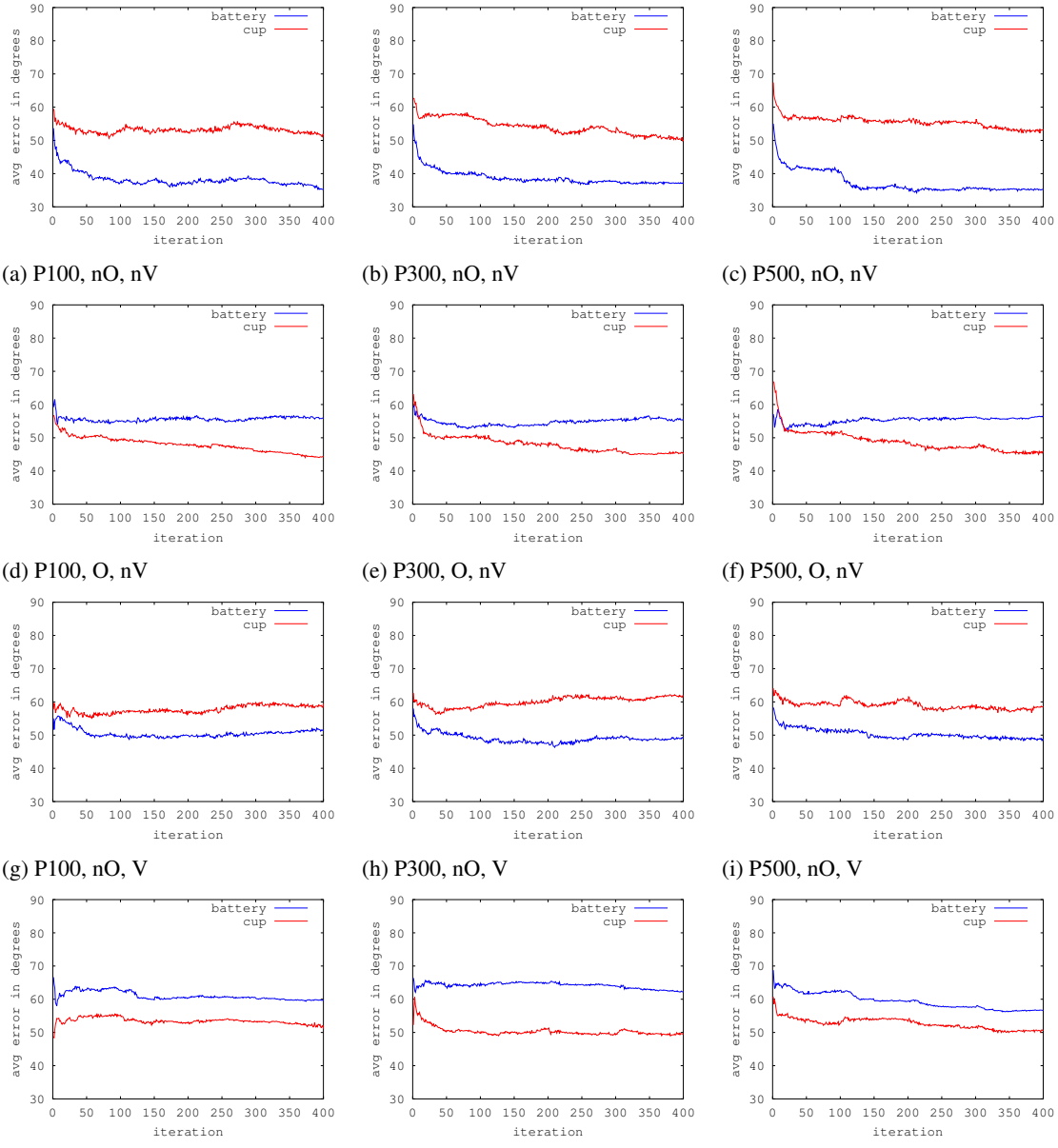


(g) P100, nO, V (h) P300, nO, V (i) P500, nO, V



(j) P100, O, V (k) P300, O, V (l) P500, O, V

Figure C.3.: Error in position for different object types. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity



(a) P100, nO, nV (b) P300, nO, nV (c) P500, nO, nV

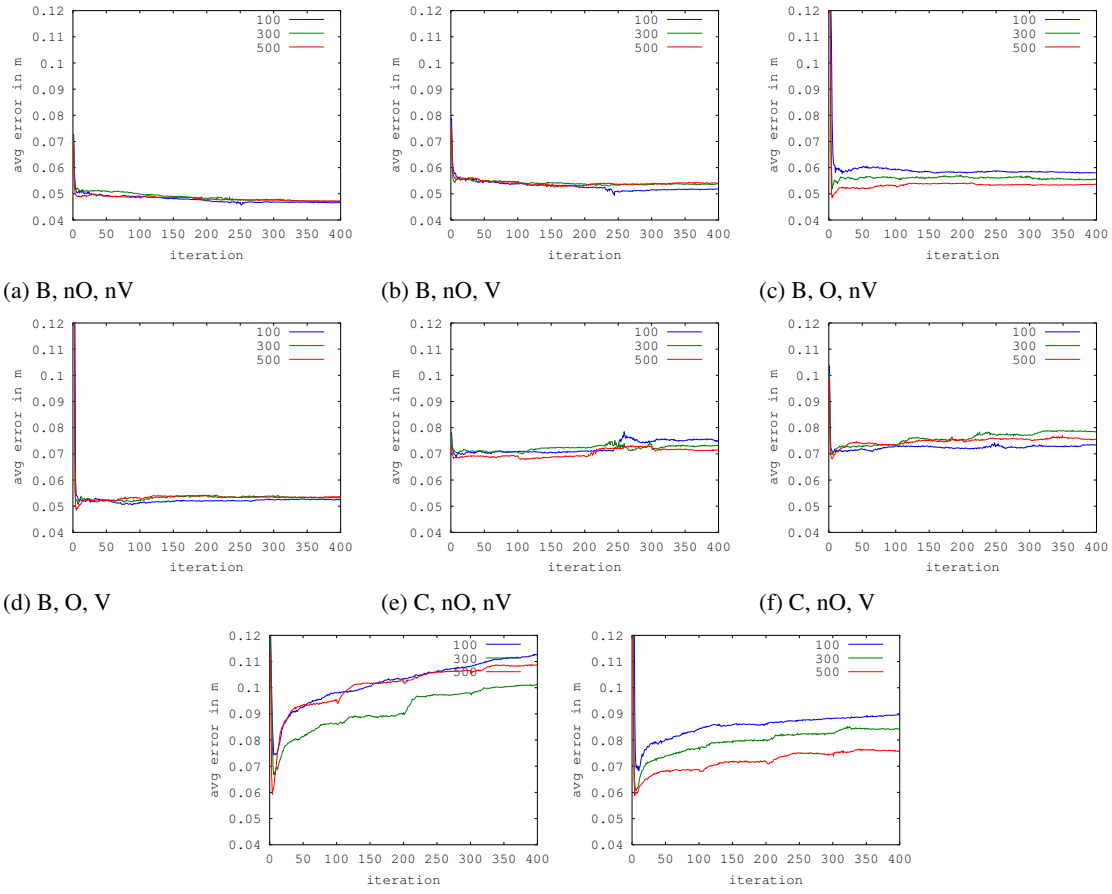
(d) P100, O, nV (e) P300, O, nV (f) P500, O, nV

(g) P100, nO, V (h) P300, nO, V (i) P500, nO, V

(j) P100, O, V (k) P300, O, V (l) P500, O, V

Figure C.4.: Error in orientation for different object types. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

C.3. Number of particles

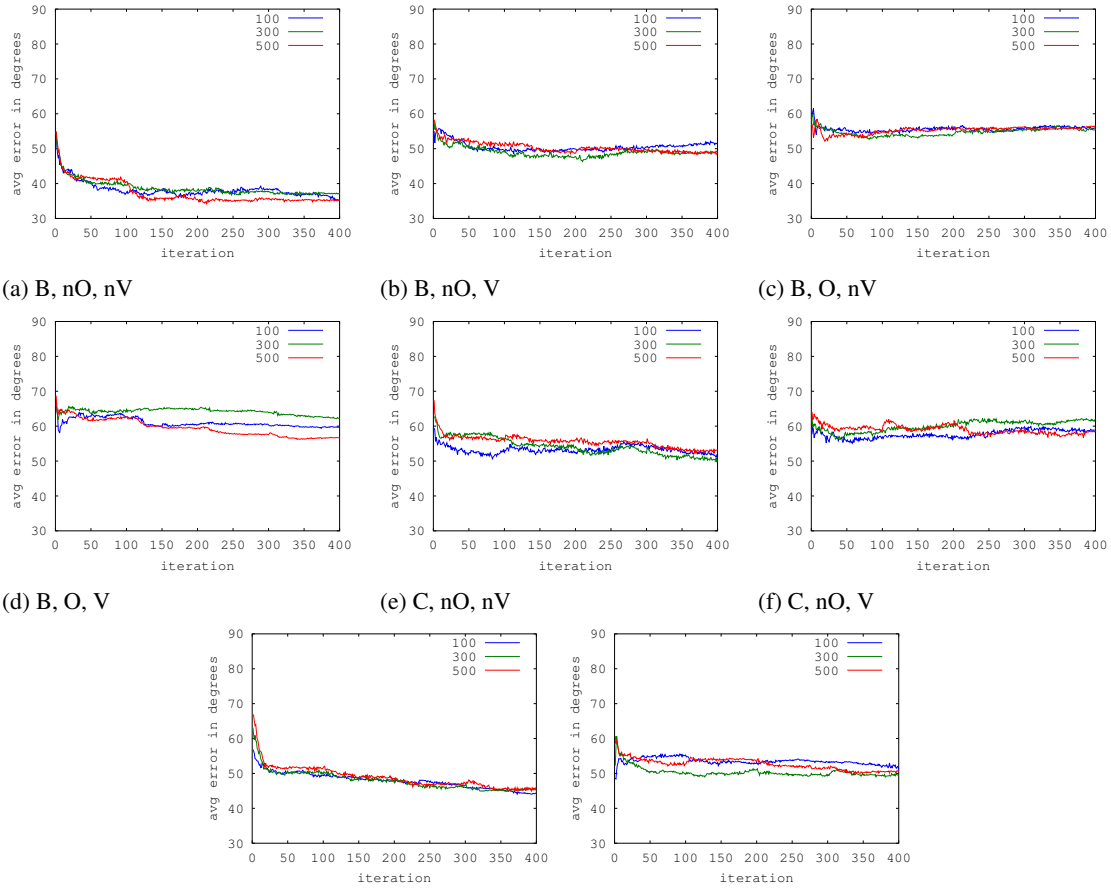


(a) B, nO, nV (b) B, nO, V (c) B, O, nV

(d) B, O, V (e) C, nO, nV (f) C, nO, V

(g) C, O, nV (h) C, O, V

Figure C.5.: Error in position for different upper bounds of particles. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity



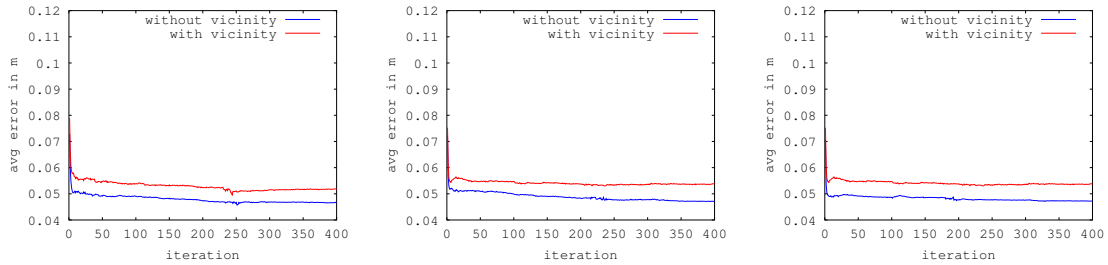
(a) B, nO, nV (b) B, nO, V (c) B, O, nV

(d) B, O, V (e) C, nO, nV (f) C, nO, V

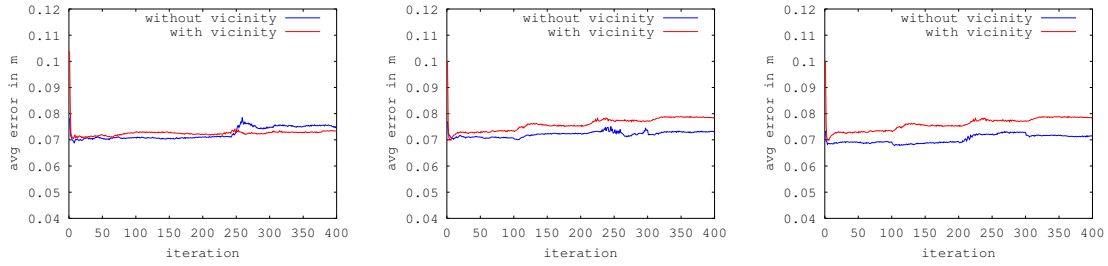
(g) C, O, nV (h) C, O, V

Figure C.6.: Error in orientation for different upper bounds of particles. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

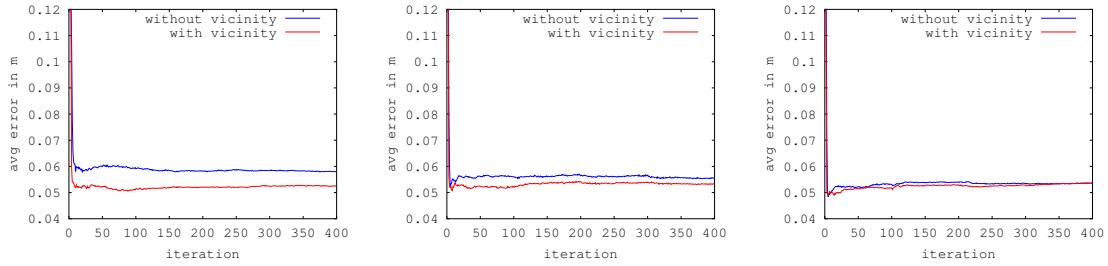
C.4. Vicinity



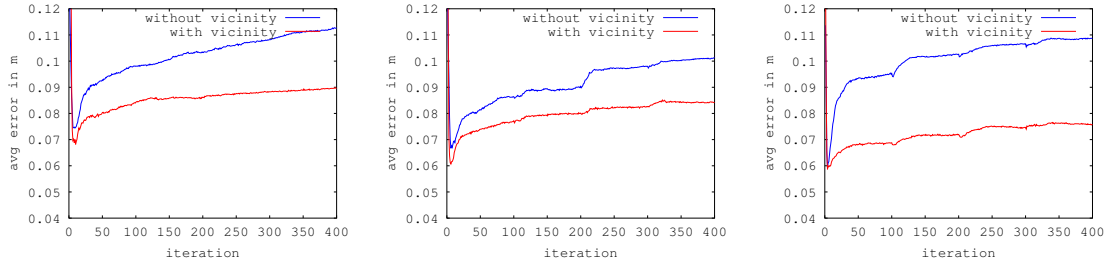
(a) B, P100, nO (b) B, P300, nO (c) B, P500, nO



(d) C, P100, nO (e) C, P300, nO (f) C, P500, nO

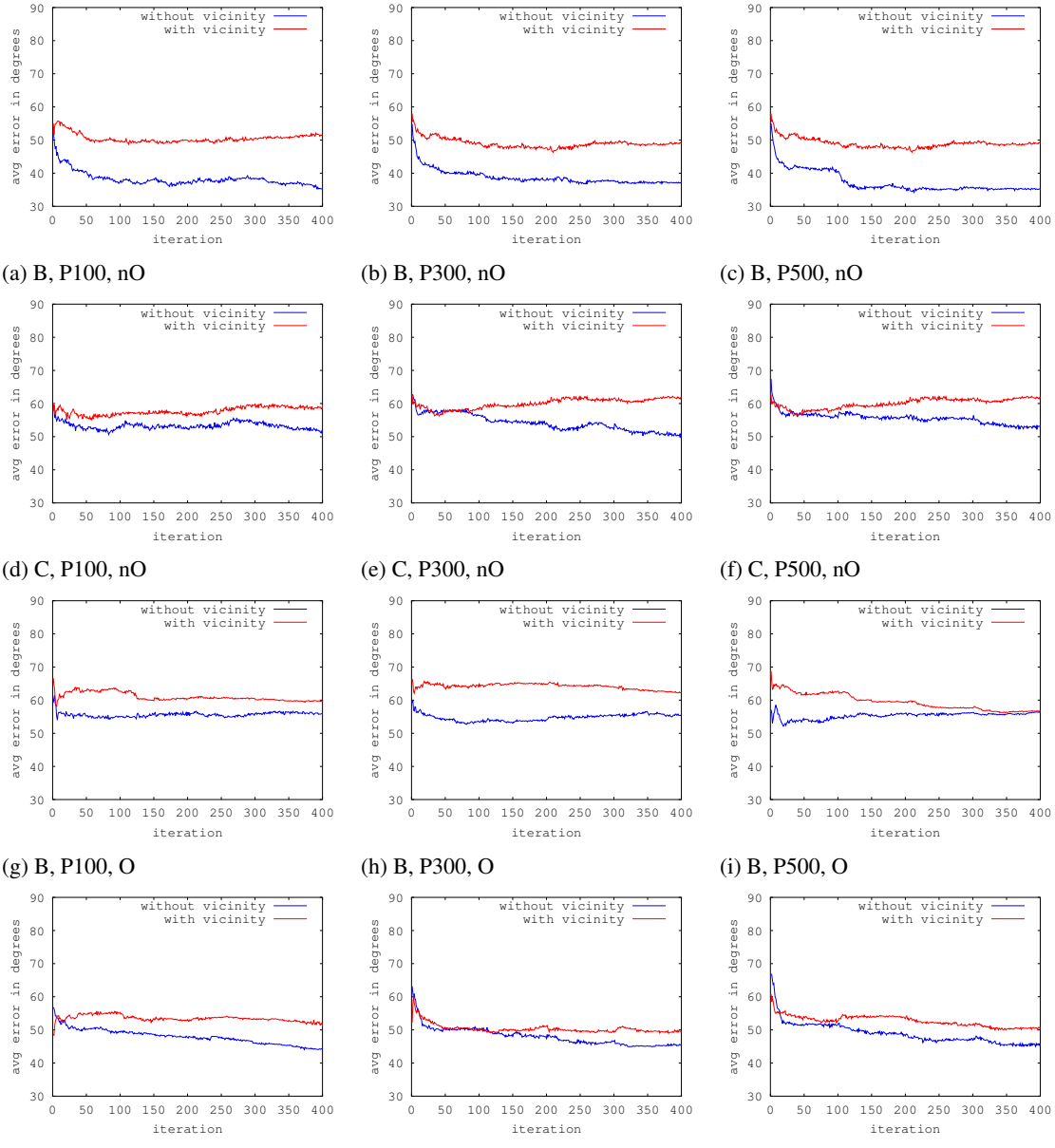


(g) B, P100, O (h) B, P300, O (i) B, P500, O



(j) C, P100, O (k) C, P300, O (l) C, P500, O

Figure C.7.: Error in position for consideration/ no consideration of the vicinity. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity



(a) B, P100, nO (b) B, P300, nO (c) B, P500, nO

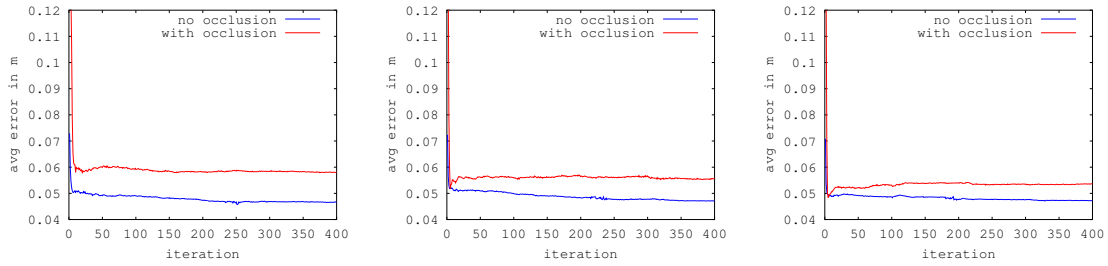
(d) C, P100, nO (e) C, P300, nO (f) C, P500, nO

(g) B, P100, O (h) B, P300, O (i) B, P500, O

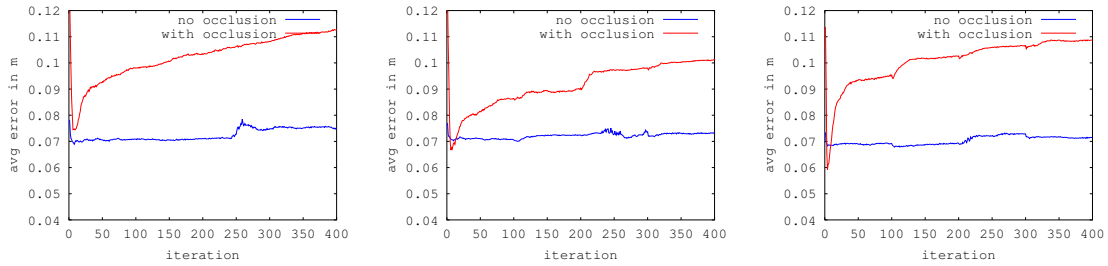
(j) C, P100, O (k) C, P300, O (l) C, P500, O

Figure C.8.: Error in orientation for consideration /no consideration of the vicinity. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity

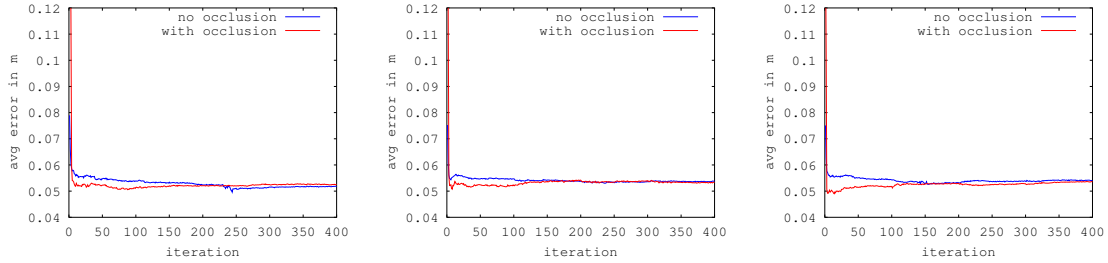
C.5. Occlusion



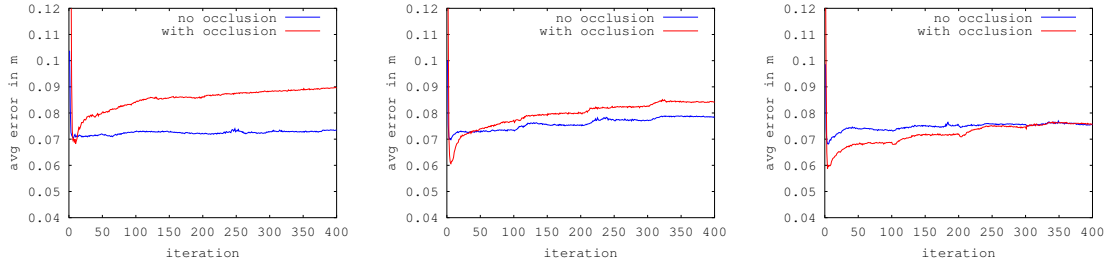
(a) B, P100, nV (b) B, P300, nV (c) B, P500, nV



(d) C, P100, nV (e) C, P300, nV (f) C, P500, nV

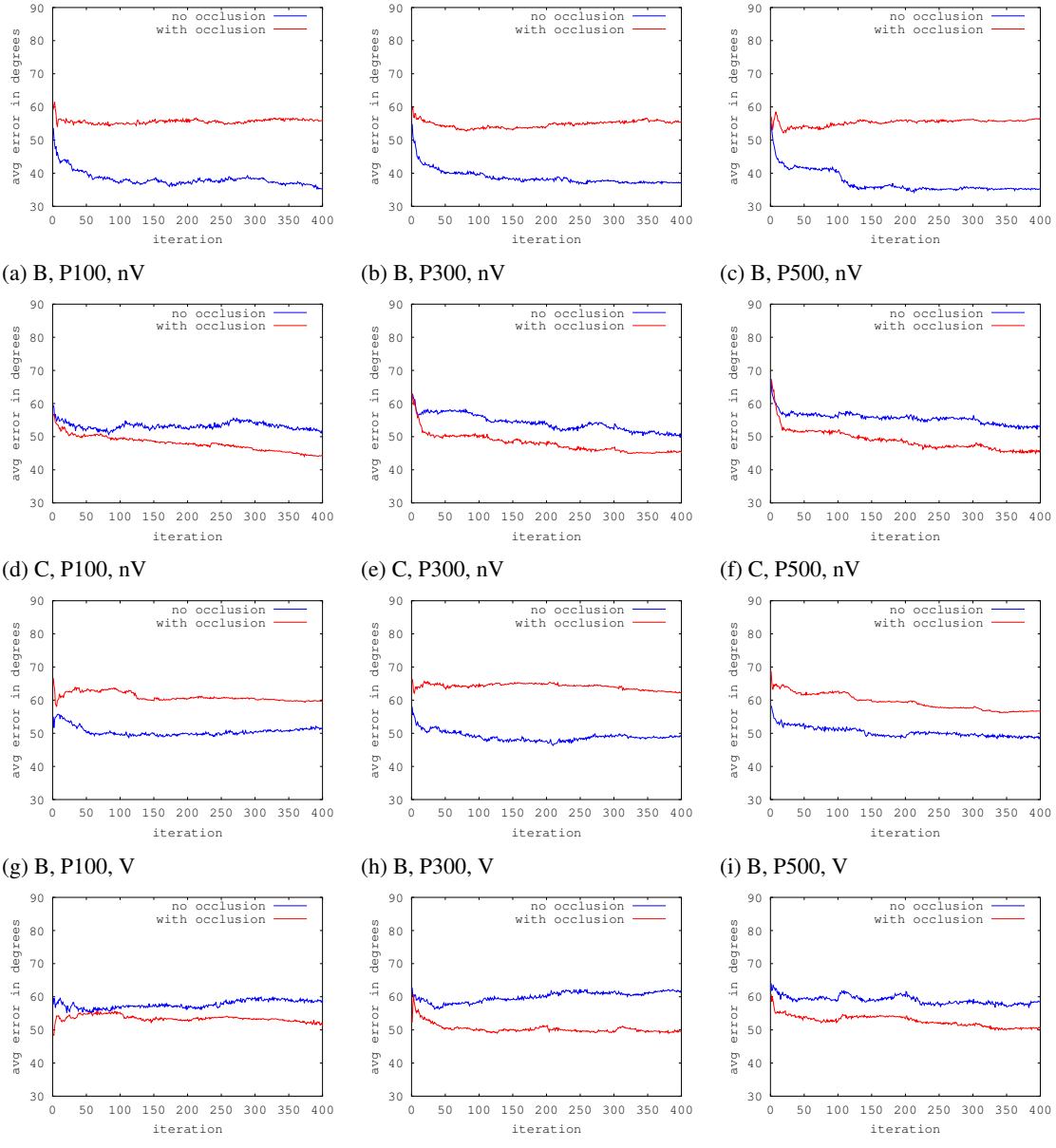


(g) B, P100, V (h) B, P300, V (i) B, P500, V



(j) C, P100, V (k) C, P300, V (l) C, P500, V

Figure C.9.: Error in position for occlusion/no occlusion of the objects. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity



(a) B, P100, nV (b) B, P300, nV (c) B, P500, nV

(d) C, P100, nV (e) C, P300, nV (f) C, P500, nV

(g) B, P100, V (h) B, P300, V (i) B, P500, V

(j) C, P100, V (k) C, P300, V (l) C, P500, V

Figure C.10.: Error in orientation for occlusion/no occlusion of the objects. B = battery, C = cup, P100 = 100 particles, P300 = 300 particles, P500 = 500 particles, nO = no occlusion, O = occlusion, nV = no vicinity, V = vicinity