Bachelor Thesis

Multiple microrobot and microobject localization and recognition

Stefan Isler

Dr. Roel Pieters

Adviser

Prof. Dr. Bradley J. Nelson

Institute of Robotics and Intelligent Systems

Swiss Federal Institute of Technology Zurich (ETH)

2014-09

Abstract

This thesis presents a localization and recognition framework for multiple micro-sized objects that is designed to be easily extendable to new object types. Known methods from the computer vision and machine learning community are employed to locate, classify and track a set of objects in a video stream. As this process is to be applied to real-time applications (30 fps), simple thresholding is used for localization, while sets of binary robust invariant scalable keypoints (BRISK) obtained from locations computed using the features from accelerated segment test (FAST) method are used for recognition.

These methods are integrated in a framework that allows for user-defined image preprocessing based on an extendable set of tunable filters, object detection using a set of very basic, unspecific object parameters such as area and contour length, and automated training of boosting classifiers for type recognition using the keypoint descriptor sets of the object types chosen by the user. New object types can be created at runtime and are characterized by a choice of keypoint descriptor sets that describe the visual appearance along with the choice of a module from an extendable list which defines the dynamics and state filters that are employed for state prediction and estimation. Configurations and type settings can easily be saved, shared and edited using extensible markup language (xml) files and a graphical user interface.

Zusammenfassung

Diese Arbeit präsentiert ein Framework zur Lokalisierung, Erkennung, Verfolgung und Klassifizierung von mehreren Mikroobjekten in aufeinanderfolgenden Bildern einer Videoquelle, das auf einfache Erweiterung auf neue Typen ausgerichtet ist. Hierzu verwendet werden bekannte Methoden aus der computergestützten Bildverarbeitung und Algorithmen für maschinelles Lernen. Da alle notwendigen Schritte in Echtzeit bei 30 bps ausgeführt werden sollen, wird simples Thresholding zur Lokalisierung angewandt. Die Charakterisierung unterschiedlicher Objektklassen erfolgt über Bildpunktbeschreibungsdatensätze, die der *binary robust invariant scalable keypoints* (BRISK) - Algorithmus für mit der *features from accelerated segment test* (FAST) - Methode berechnete Punkte erstellt.
Dies wurde integriert in ein Framework, das benutzerdefinierte Bildverarbeitungsroutinen zur Datenvorbehandlung, basierend auf einem erweiterbaren Satz von konfigurierbaren Filtern, Objekterkennung unter Einbeziehung grundlegender Eigenschaften wie Fläche und Konturlänge, sowie automatisiertes Lernen eines *boosting* - Klassifikators zur Typenerkennung unter der Verwendung der Datensätze der vom Benutzer ausgewählten Objekttypen ermöglicht. Neue Objekttypen können zur Laufzeit erstellt werden, wobei ihre visuelle Erscheinung durch eine Auswahl an Bildpunktbeschreibungsdatensätzen definiert und ihr dynamisches Verhalten, sowie der zu verwendende Zustandsfilter für Vorhersage und Abschätzung von Parametern über die Auswahl eines Dynamikmoduls festgelegt wird. Programm- und Objektkonfigurationen können gesichert, geteilt und verändert werden über *extensible markup language* (xml) - Dateien und eine grafische Benutzeroberfläche.

Contents

List of Tables

List of Figures

Nomenclature

Notation

Scalars are written in lower case letters (a), vectors in lower case bold letters ($\mathbf{a}$), and matrices in upper case bold letters ($\mathbf{A}$).

Acronyms and Abbreviations

ABF	Artificial Bacterial Flagella
bps	Bilder pro Sekunde
BRISK	Binary Robust Invariant Scalable Keypoints
EMA	Exponential Moving Average
ETH	Eidgenössische Technische Hochschule
FAST	Features from Accelerated Segment Test
fps	frames per second
FREAK	Fast Retina Keypoint
GUI	Graphical User Interface
IC	Integrated Circuit
IRIS	Institute of Robotics and Intelligent Systems
MEMS	Microelectromechanical Systems
MOLAR	Multiple Object Localization and Recognition
MSRL	Multi-Scale Robotics Lab
QML	Qt Meta Language
ROI	Region of Interest
ROS	Robot Operating System

SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
SVM	Support Vector Machines
XML	Extensible Markup Language

1 Introduction

Engineering sub-millimetre to nanoscale objects functionalized for specified tasks that can be controlled reliably is a challenging problem which is gaining a lot of attention by research groups worldwide.

1.1 Microscale robot systems

A plethora of microrobots with different shapes, actuation methods and applications have been proposed during the last years. Employing sophisticated and matured microfabrication methods also used for integrated circuits (IC) and microelectromechanical systems (MEMS), a wide variety of designs and functionalizations are feasible.

Possible actuation methods include externally applied electrical and magnetic forces, as for artificial bacterial flagella, acoustic radiation force (as used by Andrade *et al.* [5]), optical tweezers (see Curtis *et al.* [13]) and the use of microorganisms as was shown by Takahashi *et al.* [33] or Martel *et al.* [24].

1.1.1 Artificial bacterial flagella

One type of microrobot that has been studied extensively and which will be seen again later on in this thesis are artificial bacterial flagella (ABF). Artificial bacterial flagella are magnetic helical microrobots that use a cork-screw strategy for self-propulsion which is inspired by bacteria such as Escherichia coli. ABFs suspended in a low Reynolds number solution align with the direction of an externally applied magnetic field and rotate by following the rotation of the field. This rotation causes the robot to move forward in the aforementioned cork-screw-like manner, translating the rotational into a translational motion. ABFs can be used to manipulate cellular and sub-cellular objects by direct pushing [27], for non-contact methods creating specific flows in the peripheral liquid [26] or functionalized for drug release applications [29]. An ABF is depicted in Figure 1.

1.2 Microscale robot applications

Microrobotic agents have a wide range of applications where their small size promises or already shows to be highly beneficial, with two notable areas being

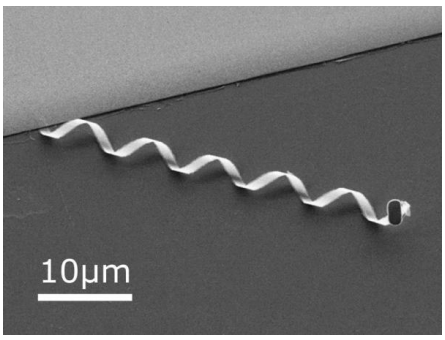


Figure 1: Artificial bacterial flagella [36].

medical applications, e.g. drug delivery or microsurgery, and the manipulation of microsized objects.

A big advantage for medical applications is that micro- and especially nanorobots can move almost freely through the human body using its natural transport system, the bloodstream. They are thus able reach essentially arbitrary locations, rendering invasive and side-effect prone techniques such as the surgical cutting of tissues unnecessary. This also allows for the delivery of drugs to the location where they are needed, e.g. a cancer site, creating the necessary drug concentrations locally instead of flooding the whole body. Nelson *et al.* give an overview of biomedical applications for microrobots [25].

In the case of micromanipulation the usage of microrobots has been shown as an eligible alternative to other more established approaches such as optical or magnetic tweezers and microgrippers.

1.2.1 X-ray crystallography

One such area where a microrobot has been used for manipulation of microobjects is X-ray crystallography. In X-ray crystallography the atomic and molecular structure of a crystal is investigated by shooting X-rays towards a sample and measuring angles and intensities of diffracted beams [35]. It is the most widely applied method for characterizing the atomic structure of new materials and has recently gained a lot of attention for the analysis of organic and biological molecules, such as proteins [17].

Tung *et al.* state that a major bottleneck of automated, high-throughput crystallography lies in the harvesting of the crystals from crystallization solutions and their preparation for exposure to X-rays [34]. This task is usually done manually

due to the small scale and delicate nature of the crystals.
While other methods to automate the crystal harvesting process have been presented, e.g. the use of optical tweezers and robotic arms, Tung *et al.* propose the use of a microrobot for noncontact manipulation. Pieters *et al.* show how this microrobot, which they named RodBot, can be used for automated crystal harvesting [28]. The RodBot is actuated by an externally applied, rotating magnetic field and rolls on the surface. If it is placed in a liquid, its motion generates a rising flow in front of it and a vortex above it. It can thus lift protein crystals off the substrate, trap them in the vortex and transport them to and deposit them onto an extraction-loop. The process is monitored and controlled with a visual control system using a camera mounted on top of a microscope that provides imagery as visible in Figure 2.

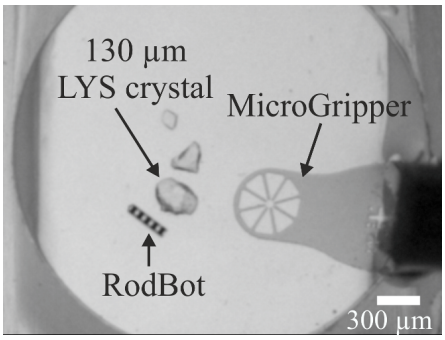


Figure 2: Microscopic view of the crystal harvesting process. Visible are the crystals, the RodBot and the extraction-tool (MicroGripper) [28].

1.3 Limitations of microrobots

The scale of microrobots introduces several difficulties as established techniques and devices known from macroscale robotics scale down unfavourably, e.g. power sources. Microrobots presented in literature usually have no onboard computational power at all, no communication means, no intrinsic power source, and rely exclusively on external control by a central unit that monitors the global state of the system. In fact, they are often not much more than mere particles that are designed to have certain properties which allow an external control unit to influence and move them by applying corresponding forces. Furthermore, their size is not only an advantage for many applications as shown earlier, it may also

represent a significant disadvantage. The actions performed by one single microrobot will have little effect and either a group of agents has to be employed or the agent has to carry out its task repeatedly over an elongated period to achieve a given goal.

1.4 Multiple microrobot systems

For some applications multiple microrobot systems are the only feasible solution, e.g. drug delivery, since the cargo of a single robot is very limited. For real applications it is therefore often crucial to steer a large group of agents in a controlled manner. This is a challenging problem because of the fact that many actuation methods offer only one input, i.e. all agents receive the same input signal.
One solution is the use of heterogenous swarms: Microrobots of the same type can be developed such that they differ in certain properties, e.g. magnetization or size, and thus react differently to the same input field. Such an approach was chosen by Donald *et al.* [16] and Diller *et al.* [15].
Another approach is the use of specialized motion planning algorithms that under certain conditions drive uniform microrobots to a target position with the use of only one input signal, as is shown by Becker *et al.* [7].

1.5 This thesis

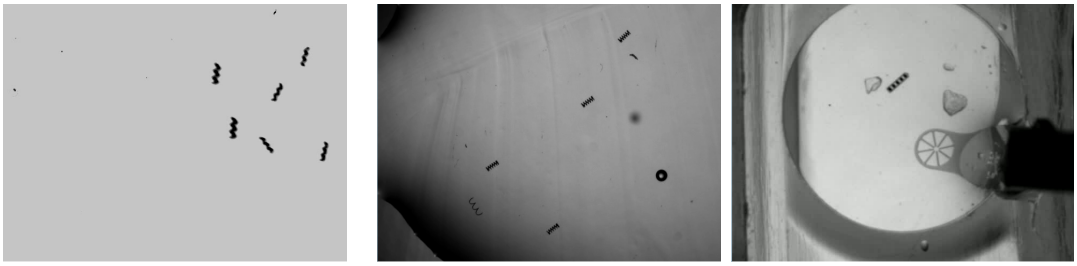
In order to enable a centralized control unit to control multiple microobjects, the first step is to detect, track and also classify these objects if there is more than one type present in the scene (e.g. crystallography, where an automated program routine will have to distinguish between the microrobot and the crystals). Not only state prediction but also motion planning in future projects will be type dependent.

This thesis presents a framework that tackles these problems in real time (30 frames per second) on the test platform (Intel i7 at 2.93 GHz running Windows 7). It is easily configurable by the user and assumes as few prerequisites as possible, hopefully enabling it to be used for different applications as well as for different types of robots. The used libraries support other platforms like Linux or Mac as well.

First the three basic stages in the program flow, *localization of objects*, *object matching and tracking* and *classification and type recognition* are being described in Chapters 2, 3 and 4 respectively, then their integration in the overall program architecture is presented along with its layout and interface in Section 5. Finally, Chapters 6 and 7 conclude the report with an outlook and a summary of the project.

2 Localization of objects

The first step in the processing pipeline is the localization of all objects in the scene that are of interest. A scene is represented at each instant in time by an image of a video stream. The video stream source may either be a video file or a camera. The primary model case for this thesis was a video with a group of six heterogenous ABF microrobots moving in liquid, which is illuminated to appear as an almost homogenous grey background (Figure 3a). Secondary models were a video file with two different object types, a thin and a thick helix (Figure 3b), and a video file of crystal harvesting with a RodBot (Figure 3c).



(a) Primary model case (b) Secondary model case 1 (c) Secondary model case 2

Figure 3: Images from the video files that were used as model cases for the framework.

Such videos are obtained by mounting a camera on a microscope observing the scene from a top-view position, e.g. a PointGrey Grasshopper 50S5C CCD Firewire Camera fitted to an Olympus SZX-9 microscope. In all three model cases the videos had a resolution of 640x480 px at 8 bits color depth. The implemented algorithms are however not resolution specific and will work for arbitrary settings. All computer vision algorithms have been implemented using methods of the open computer vision library (OpenCV). Refer to [1] or [8] for more information about the library.

2.1 Solution approach considerations and task constraints

The established method for segmenting images and the extraction of object features is generally speaking a three step procedure. In a first step image filters are applied that aim at normalizing the image and prepare it for the second step, in which the image is segmented in interest and non-interest parts using threshold-

ing. The interest parts are then extracted and quantified in the third step [32]. General object parameters to be calculated are the position and angle as well as the size.

2.1.1 Timing

Filters applied in this first phase of the process flow (subsequently called *preprocessing phase*) are especially time-consuming and must be chosen with care because each filter will have to process every point in the image at least once. In the most simple case and for a resolution of 640x480 px as in the model configurations a total of 307,200 pixels must be processed. On a single processor unit running at 3 GHz even with only atomic operations[1] – which are seldom sufficient – this results in a minimal execution time of about 0.1 ms. Sophisticated filters using convolution and other techniques require a multiplicity of this time to be carried out and execution times larger than 100 ms are often being observed. Such solutions are not feasible for real-time applications running at 30 Hz.
This problem is often tackled by processing the whole image only in a first step and afterwards using the position of the detected objects to calculate regions of interests (ROI) which are the only regions processed by the algorithms in future steps. Such an approach can bring down computation times significantly and is especially advisable in cases where it is known a priori that only one object will occur in the scene or no objects will leave or enter the scene during execution. In this thesis however, such assumptions are not made. This basically leaves three options:

- The whole image is being processed at every time step, limiting the possibilities in the preprocessing stage to the most vital steps only.
- Regions of interest are being calculated and processed, with the whole image being processed periodically to spot new objects. This will lead to periodic execution time peaks, which is only acceptable if in the time between two peaks sufficient time overhead is given to compensate for the expected periodic timing violations.

[1] Atomic operations are operations that can be carried out by a CPU in a single instruction.

- Additionally to the regions of interest given by detected objects four additional, fixed regions of interest could be defined to observe the border area where any new object necessarily occurs first. This method might run into problems if objects newly entering the scene are masked by known objects occupying the border position where they enter.

In this thesis the first approach was chosen, accepting the challenge of limiting the preprocessor filter section. This decision is also in agreement with the possibility given to the user to have considerable manual control over the preprocessing filter stage, as described later.

2.1.2 Extraction of object features

For extraction of object positions the use of feature descriptors was first considered. Feature descriptors such as *scale-invariant feature transform* (SIFT) which was proposed by Lowe [23] have been shown to successfully locate different objects in complex settings. However, test runs on the model cases have shown that the expected sources provide not enough resolution and detail for secure object matching and that furthermore the few keypoints obtained per object (between 2 and 20 keypoints per object were observed) are insufficient to extract any reliable object data, e.g. position or orientation of robots. Figure 4 shows keypoints

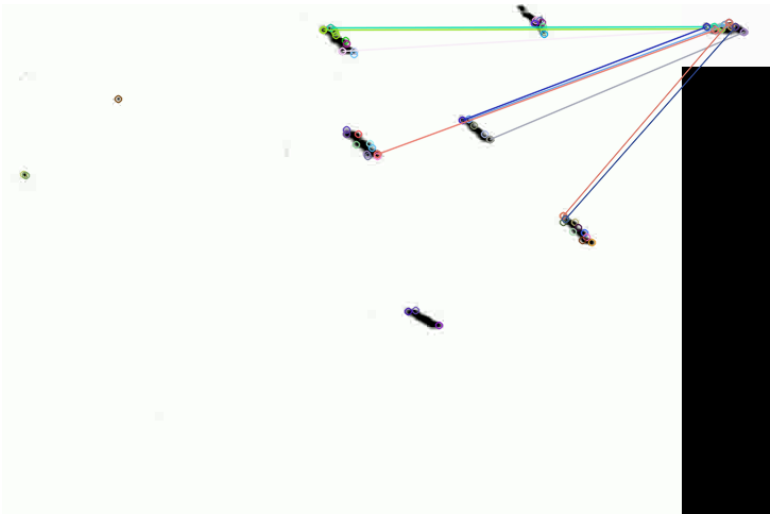


Figure 4: The colored circles depict FAST keypoints while the lines represent matches using BRISK feature descriptors and brute force matching.

detected for model case 1 using *features from accelerated segment test* (FAST)

[30] and matching using *binary robust invariant scalable keypoints* (BRISK) [22] descriptors.

While feature points of the robots can in many cases be sufficiently matched as being feature points of a specific robot, they cannot be used to extract the object properties in which we are interested in at this stage of the algorithm. They were however used for classification purposes and are therefore discussed in more detail in Chapter 4. For object localization it was decided to resort to a classical thresholding technique.

2.2 Image normalization and preparation for thresholding

Before suitable methods for image normalization and/or preparation for thresholding can be identified, the targeted image properties need to be defined. The optimal input for any threshold operation is an image where the interest regions (black in this thesis) are already clearly separated from any non-interesting region, also called the *background* (white).

First a few basic observations and conclusions are made:

1. Input streams are grey level only and it is thus sufficient to operate on one-channel, intensity images.

2. Video streams and experimental settings can be created such that lighting is almost uniform and interest objects are in clear contrast to the background, facilitating preprocessing. This observation was part of the reasoning why full image processing at 30 Hz was chosen. The experimental setup is thus already a crucial part if real-time processing is targeted. An insufficient experimental setup not only complicates object recognition, it also corrupts extracted data through unnecessary introduced noise.

3. Not all areas in the image stream are of interest. In the crystal harvesting stream (secondary model case 2) for instance, only the work area inside the petri dish where crystals, robot and the extraction-loop are located is of interest. Processing may thus be limited to a user defined area in the image from the beginning. This option was implemented in the framework.

4. Since the whole framework is targeted at multiple varying setups, it is unreasonable to expect the existence of one single filter performing optimally in all cases and fulfilling the timing constraints. For preprocessing different applications require different approaches and new applications will most likely require the user to define new, basic preprocessing steps. It is thus necessary to create a dynamic preprocessing stage where defined algorithms easily can be added and adjusted at runtime and for which new algorithms may be readily designed if necessary. A user may also choose to use a more sophisticated preprocessing stage in cases where real-time processing is no requirement.

5. The time consumption of image filters needs to be compared with their benefits. E.g. Gaussian filtering and other smoothing operations such as bilateral filtering are often used as the first stage in image processing to increase robustness and reduce the influence of noise [32]. These operations have however shown to be little beneficial for tracking and detection in the model cases observed and were thus not considered.

6. Test runs show that filters requiring the calculation of image statistics, e.g. histograms, fail to meet the time constraint if the statistics are recalculated for every image frame. However, since many image properties such as lighting are approximately static for any given stream, one option is to calculate the required statistics only for the first frame and use the results for all subsequent frames. Violating the timing constraint only for the first frame is acceptable and leads to no additional problems if enough time overhead is present in future time steps. Enough overhead is highly advisable for real-time applications since extraordinary computation peaks can always occur, especially on systems not built for real time processing.

2.2.1 Color range expansion

For model case 1 and other cases where dark interest objects are displayed on an approximately uniform bright background a simple algorithm was developed which automatically expands the color space occupied to span the whole range from black (0) to white (255). For the very first frame the histogram is calculated from which the darkest occuring color as well as the brightest occuring peak are

extracted, using a peak detection algorithm (shown partly in Algorithm 1) that calculates a set of histogram peaks from which basically the brightest peak is chosen. With these two values the transformation function is given by:

$$c_{out} = \frac{255}{(c_b - c_d)} * (c_{in} - c_d) \tag{2.1}$$

where c_{out} is the output color of the pixel, c_{in} the input color, c_d the darkest color occuring in the image and c_b the color of the calculated, brightest peak. Since the image statistics are calculated only for the first frame, these input-output color-pairs are fixed and can be saved in a lookup table (LUT)[2] in order to speed up future processing. Negative values are set to zero. Figure 5 shows a typical histogram for model case 1 with detected peaks, as well as an exemplary input and output picture of the algorithm.
Test runs show how the algorithm performance benefits from the use of a lookup table (see Table 1).

Table 1: Color range expansion algorithm performance.

Test case description	**Average comp. time**
Without LUT, first step with histogram calculation	19.5842 ± 1.0030 ms
Without LUT, following time steps	13.1662 ± 1.4889 ms
With LUT, first step with histogram calculation	11.5237 ± 0.7311 ms
With LUT, following time steps	4.2924 ± 0.4630 ms

2.2.2 Brightness and contrast adjustment

The most important and basic task of the preprocessing stage is brightness and contrast adjustment. As a tool for user-controlled preprocessing a simple brightness and contrast adjustment filter was implemented with user defined contrast factor k and brightness constant b. For performance a lookup table is filled with values according to the transformation law

$$c_{out} = k * c_{in} + b. \tag{2.2}$$

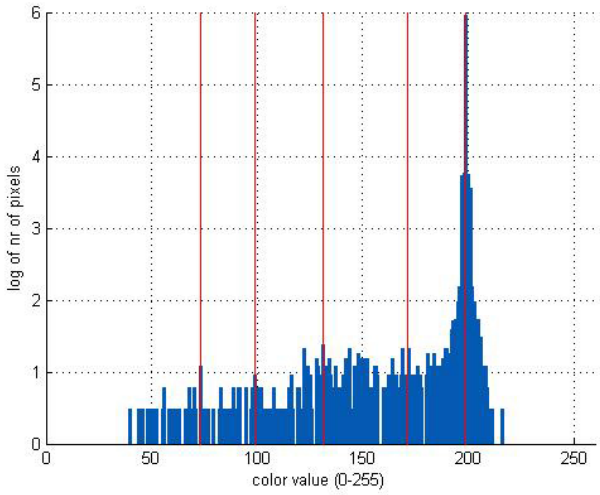
[2]Refer to `http://en.wikipedia.org/wiki/Lookup_table` for more information about lookup tables.

Algorithm 1 Histogram peak calculation

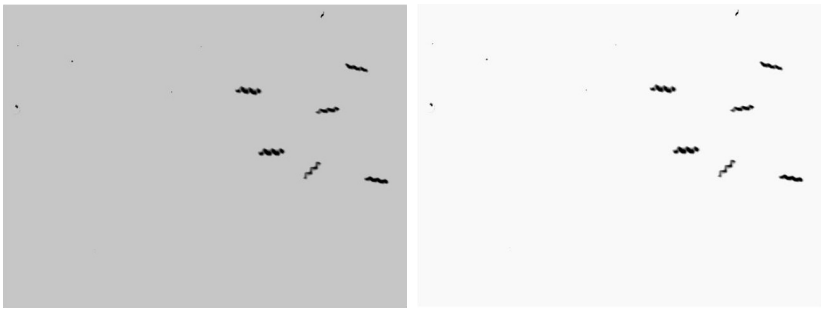
Input: vector *hist* with histogram counts for the 256 color values
Input: peak exclusiveness width *exclwidth*
Output: vector with color peaks *peaks*

$currentPeak \leftarrow hist_0$
$peakCandidate \leftarrow 0$
$edgeWidthCount \leftarrow 0$
$lastSameHeightPeak \leftarrow -1$
$minEdgeWidth \leftarrow exclwidth/2$

for all $hist_i$ **do**
  if $hist_i < currentPeak$ **then**
    $peakCandidate \leftarrow i$
    $currentPeak \leftarrow hist_i$
    $edgeWidthCount \leftarrow 0$
    $lastSameHeightPeak \leftarrow -1$
  else if $hist_i = currentPeak$ **then**
    $lastSameHeightPeak \leftarrow i$
    $edgeWidthCount \leftarrow 0$
  else
    $edgeWidthCount++$
    if $edgeWidthCount = minEdgeWidth$ **then**
      if $lastSameHeightPeak = -1$ **then**
        $peaks \leftarrow (peakCandidate, currentPeak)$
      else
        $peaks \leftarrow (\frac{peakCandidate+lastSameHeightPeak}{2}, currentPeak)$
      end if
      $peakCandidate \leftarrow -1$
      $currentPeak \leftarrow 0$
    end if
  end if
end for



(a) Peak detection and image histogram



(b) Input image (c) Output Image

Figure 5: Different stages of the color expansion algorithm.

2.3 Thresholding

In order to segment the preprocessed input frame in interest areas (representing object candidates) and background each pixel color of the gray scale image (0...255) is compared to a threshold value and then assigned either color black or white depending on the result. The threshold value can either be a fix, global value or a variable value which is calculated based on the surrounding area of the pixel. The latter method is called *adaptive thresholding* and is especially beneficial in cases of nonuniform lighting such as given in secondary model 1. For these kinds of settings it can proof very challenging to find a global threshold value which is still able to separate the image satisfactorily.

However, adaptive thresholding comes at the cost of increased computation

times and the method of choice must thus be to ensure uniform lighting conditions through the experimental setup and use a global threshold to segment the images whenever possible.
Such a global threshold can be calculated automatically quite easily for images as in the primary model case, e.g. the median of the darkest color peak and the brightest color peak. Since switching through a few values manually is done fast, allows for custom adjustments and no automatic threshold value calculation routine seems applicable to cases such as the secondary model 1 (without the use of more sophisticated and time-consuming algorithms), no automation method was implemented and setting the threshold value was left to the user. The standard global threshold is set at 220, which has proven a reasonable value for the primary model case. Furthermore, it was shown that a global threshold value is even able to satisfyingly segment nonuniform lighting conditions as in secondary model case 1, given a clever preprocessing filter setup. The output of the thresholding stage is depicted in Figure 6a.

One solution which could be interesting to be implemented in an improved future approach is the use of an adaptive threshold calculation along with a lookup table which either saves a specific threshold value for each pixel in the image or a specific threshold value for predefined regions (quads) in the image. Since lighting conditions can be assumed static such a solution most likely would improve performance under nonuniform conditions while only adding to the computation time of the first frame – which can be regarded as a calibration phase.

2.4 Extraction of object features

2.4.1 Basic localization and filtering of the object set

Objects are located in the segmented and inverted binary images using OpenCV's findContours() method, which is based on the border following algorithm of Suzuki *et al.* [31] (see Figure 6b). While further filtering of the extracted objects is not necessary since the framework is designed to handle multiple objects, this option is still given to the user who can set a minimal and maximal area for objects at runtime. Additionally a minimal contour length constant can be used to override the minimal area boundary in the case of very long and thin objects. The filtering process is presented in Algorithm 2.

Algorithm 2 Object filtering

Input: vector *contours* with all contours of connected components
Input: minimal area *minArea*, maximal area *maxArea* and minimal contour length *minContour*
Output: vector *objects* with the contours of all objects that will be used for further processing

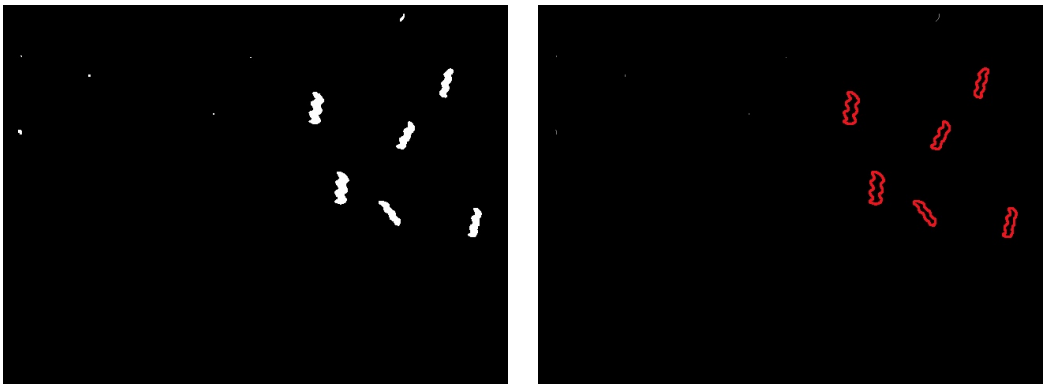
for all $contours_i$ **do**
  calculate *contourArea*
  calculate *contourLength*
  if $(contourArea > minArea || contourLength > minContour)$ & $contourArea < maxArea$ **then**
    $objects \leftarrow contour_i$
  end if
end for

2.4.2 Intermediate entities

As each frame is being analyzed several intermediate entities are calculated and kept in order to be provided to subsequent process steps, e.g. allowing objects to perform additional property calculations based on image data. For each contour two types of minimal rectangles are created, upright (Figure 6c) and rotated, using OpenCV's boundingRect() and minAreaRect() methods respectively. The upright images, which represent regions of interest, are used to segment the frame in subimages on which all further image processing is executed. For image moment calculation a set of inverted subimages is being created (Figure 6d). See appendix A.1 for a data flow diagram of the object extraction stage.

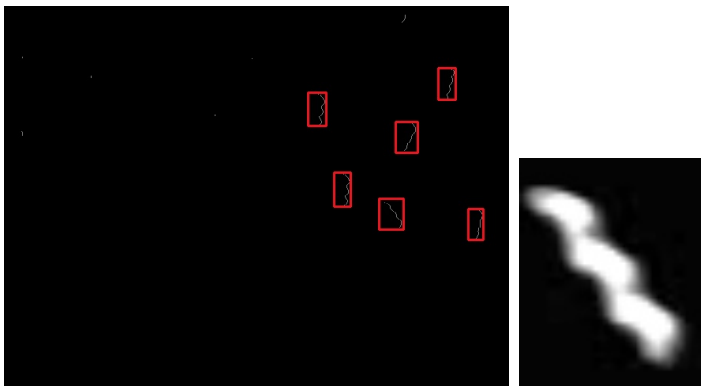
2.4.3 Image objects state calculation

The framework extracts the basic raw state properties position (x,y), angle and area for each detected object using image moments. Values obtained through this method are expected to be more accurate than calculations based on the contours. State filtering is left to the objects themselves.
Since image moments can be created only on upright rectangular images (using OpenCV's moments() method), the area not occupied by the object whose moments are being calculated needs to be masked such that no other close objects lying partly inside the currently examined ROI interfere with the calculation.



(a) Binary image after thresholding step

(b) Extracted contours of objects



(c) Upright rectangles encompassing the objects

(d) Enlarged exemplary subimage

Figure 6: Intermediate steps during object localization.

It is assumed that the objects' contours are of basically convex shape and thus can be approximated using the previously calculated rotated rectangles. This assumption is applicable also for the (intrinsically non-convex) helical shape of ABFs since their outer line – which is of interest here – is cylindric. Crossings of objects are neglected for these calculations, their influence must be treated as noise. Rectangles have the advantage that their geometry allows very fast calculations, e.g. when calculating if a point lies inside or outside their area (which is used in methods presented in the next chapter), and a rectangle class exploiting these advantages has been implemented.

After the rectangle mask is applied to the inverted image of the robot, the image moments are calculated [1]:

$$m_{ji} = \sum_{x,y} c(x,y) * x^j * y^i \tag{2.3}$$

$$mu_{ji} = \sum_{x,y} c(x,y) * (x - \bar{x})^j * (y - \bar{y}^i) \tag{2.4}$$

where m and mu are the image moments, x,y pixel positions, $c(x,y)$ the color (intensity) at the given position and $(\bar{x}, \bar{y})$ the mass center in coordinates relative to the respective subimage with

$$\bar{x} = \frac{m_{10}}{m_{00}},\ \bar{y} = \frac{m_{01}}{m_{00}}. \tag{2.5}$$

Using the image moments the object states are:

$$x_{obj} = \bar{x} + x_{img} \tag{2.6}$$

$$y_{obj} = \bar{y} + y_{img} \tag{2.7}$$

$$A_{obj} = m_{00} \tag{2.8}$$

$$\phi_{obj} = 0.5 * \tan\left(\frac{2 * mu_{11}}{mu_{20} - mu_{02}}\right) \tag{2.9}$$

with x_{obj} and y_{obj} the object coordinates in frame coordinates, x_{img} and y_{img} the position of the subimage relative to the frame and A_{obj} and ϕ_{obj} the object's area and orientation angle.

Equation 2.9 yields an ambiguous result as the correct angle might either be the one obtained or its orthogonal counterpart. In order to resolve the ambiguity the four corner points of the rotated rectangle enclosing the objects are used for a simple linefit (using OpenCV's fitline() method). The orientation of the obtained line is then calculated and the result compared to the previously calculated object orientation angle. If necessary, the angle obtained through image moments is flipped.

3 Object matching and tracking

All objects detected by the framework are instantiated as actual class objects of type SceneObject or derived classes (see Chapters 4 and 5). This allows for type dependent functions to be called by the framework.

From a matching perspective, objects are divided in three groups:

- **Active objects** are objects which were successfully located in the previous frame.
- **Missing objects** are objects known from earlier frames which could not be located for one or a few frames but are still expected to be in the scene. This group adds additional robustness to the algorithm, e.g. to sporadic lighting variations which can cause objects to become brighter for an instant and thus may cause them to disappear from the detected object set temporarily.
- **Lost objects** are objects which have been located in the past but are lost, usually because they left the observed image area. Currently no object matching over the image boundaries is performed even though some limited methods might be feasible, e.g. exploiting parallel movements of objects of the same type where one agent temporarily leaves the scene and returns after a short time.

3.1 State prediction

In order to relate objects from previous frames to objects in the current frame each state extracted from the current frame has to be compared to the predicted states of objects from previous frames. New states are accepted to belong to a known object if certain conditions are given. Otherwise they are identified as a state of an object which is unknown and thus new to the scene.
The choice of the correct state prediction algorithm depends on the dynamics of the object and is therefore type dependent. Accordingly each scene object class must have its own prediction method. The set of predicted states is obtained by iterating through all objects in the active and missing set and calling their respective prediction routine. Prediction methods may include different degrees of freedom, assuming constant acceleration between frames, moving average filters,

Kalman filters and other techniques.
The optimal state prediction algorithm exactly models the dynamics of each object in the scene, their interplay as well as uncertainties introduced by sensor and process errors. Additionally it will require knowledge about the control input of the system. In the given setting, however, not only the input is unknown but neither are the exact fluid dynamics of the liquid or the weight and exact shape of the objects. Since the only data that can be inferred by the visual tracking system are positions and (relative) sizes, no actual dynamical model based on physical forces, e.g. Newtonian, can be derived for the objects. The prediction algorithms must therefore rely solely on kinematic considerations.
In order to obtain a discrete model suitable for fast calculations each new frame is associated with an incremental time step of size 1 and timing inconsistencies between subsequent frames are either neglected or treated as process noise. Some of the algorithms used also infer a better estimate of the true state of the objects than given by the raw data.

3.1.1 Unconstrained motion

The most general sort of kinematics an object can exhibit is unconstrained motion, with the velocity in x- and y-direction and the angular velocity independent of each other. The state $\mathbf{x}$ at a time k can then be approximated by assuming constant acceleration between time steps:

$$\mathbf{x}(k) = \begin{pmatrix} x_k \\ y_k \\ \phi_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \phi_{k-1} \end{pmatrix} + \begin{pmatrix} \dot{x}_{k-1} \\ \dot{y}_{k-1} \\ \dot{\phi}_{k-1} \end{pmatrix} + 0.5 * \begin{pmatrix} \ddot{x}_{k-1} \\ \ddot{y}_{k-1} \\ \ddot{\phi}_{k-1} \end{pmatrix} \tag{3.1}$$

where

$$\dot{x}_{k-1} = x_{k-1} - x_{k-2} \tag{3.2}$$

and

$$\ddot{x}_{k-1} = \dot{x}_{k-1} - \dot{x}_{k-2} = (x_{k-1} - x_{k-2}) - (x_{k-2} - x_{k-3}) \tag{3.3}$$

with $\dot{y}$,$\dot{\phi}$,$\ddot{y}$ and $\ddot{\phi}$ calculated similarly. This model may be simplified further by the stronger assumption of constant velocities.

While prediction using such a bare model is possible, the generated predictions will be very susceptible to noise and other unmodeled influences. Thus filters

should be added in order to smooth results and add robustness. The models implemented in the framework so far include one using an *exponential moving average filter* to smooth the results as well as another that makes use of a *Kalman filter* to not only smooth predictions but also deal with uncertainties in the model and estimate the true object state.

- The **exponential moving average filter** (EMA) [10] is a computationally very efficient lowpass filter with exponentially weighted data. The interest measure m is exponentially smoothed according to the simple rule

$$m(k) = \alpha_m * m_{meas}(k) + (1 - \alpha_m) * m(k-1) \tag{3.4}$$

  where m_{meas} is the measured, raw data that is being added and α_m is the smoothing factor assigned to m. Such smoothing is applied to any of the three interest measures in (3.1) and their respective derivatives separately. The smoothed values are then used for prediction according to (3.1).

- Since the kinematic model for unconstrained motion as given in (3.1) is linear, a **Kalman filter** [21] can be applied to calculate estimates and predictions for the true state of the object, considering uncertainties in the obtained data and model. The state transition matrix follows directly from the equations and depends only on the choice of the state vector. In order to keep the computational effort at a reasonable level the velocities can be assumed constant with the accelerations treated as noise or added through an artificial input. Alternatively three separate Kalman filters may be applied for x and y position and the angle ϕ, reducing the size of the involved matrices and therefore the number of required multiplications per time step. The latter simplification is possible only if no coupling between the three states can be safely assumed. The Kalman filter furthermore requires a process and a measurement noise covariance matrix. While the process noise can only be estimated, depending on the chosen simplifications and models, the measurement noise can be approximately calculated from data where the state of the contained objects is considered static. In such a case any changes in the measurements during this time must necessarily be caused by noise that is introduced by the camera or other parts of the environment or signal path. The obtained variances were $\sigma^2_{x,y} = 0.005\ px^2$

for the position and $\sigma_\phi^2 = 0.0001\ rad^2$ for the orientation. They represent the measurement noise in situations where an object is clearly locatable and no other objects in the scene affect the measurement quality.

3.1.2 Nonholonomic motion

Microrobots like the types present in the model cases can not move freely in any direction at any time. ABFs are basically restricted to move approximately in the direction of their main axis (pointing upwards in reality to counterbalance the effect of gravity) and the RodBot moves perpendicular to its axis on the underlying surface, due to its rolling locomotion. Both can be rotated in place. Such nonholonomic systems can be modeled by a kinematic propagation law where the velocity vector is always aligned to the particular *movement axis* of the object type. That is, using a three-dimensional model

$$\mathbf{x}(k) = \begin{pmatrix} x_k \\ y_k \\ z_k \\ \phi_k \\ \theta_k \end{pmatrix} = \begin{pmatrix} x_{k-1} + (v_{k-1} + \frac{1}{2} * \dot{v}_{k-1}) * \cos\phi_{k-1} * \cos\theta_{k-1} \\ y_{k-1} + (v_{k-1} + \frac{1}{2} * \dot{v}_{k-1}) * \sin\phi_{k-1} * \cos\theta_{k-1} \\ z_{k-1} + (v_{k-1} + \frac{1}{2} * \dot{v}_{k-1}) * \sin\theta_{k-1} \\ \phi_{k-1} + \dot{\phi}_{k-1} + \frac{1}{2} * \ddot{\phi}_{k-1} \\ \theta_{k-1} + \dot{\theta}_{k-1} + \frac{1}{2} * \ddot{\theta}_{k-1} \end{pmatrix} \tag{3.5}$$

where

$$v_{k-1} = \sqrt{\dot{x}_{k-1}^2 + \dot{y}_{k-1}^2 + \dot{z}_{k-1}^2} \tag{3.6}$$

is the absolute speed of the robot at time $(k-1)$, z_k the z-coordinate perpendicular to the image plane and θ_k the angle between the robot's movement axis and the x-y-plane, i.e. the image plane. The third dimension attracts interest since rod-like robots submerged in a fluid do not only rotate and therefore change orientation in the directly observable image plane, but also in planes perpendicular to it.
However, the image stream provided by the camera mounted above the observed microobjects does not yield enough information to reliably calculate all five state variables, especially since no data about the size of the objects or the exact geometric setting of the camera is available. This issue is resolved with the following two considerations: The inaccessible z_k state is of no interest to the current task at hand and can be disregarded. The θ_k state on the other hand is of much interest since it has not only an influence on the observable speed of the

robot and thus the predicted position, but also describes the second orientational degree of freedom. It could be approximately calculated through the currently observed robot length l_k if the true absolute length l_{abs} and the true absolute diameter d were both known, given that

$$\begin{aligned} l_k &= l_{abs} * \cos\theta_k + d * \sin\theta_k \\ &= \sqrt{l_{abs}^2 + d^2} * \sin\left(\theta_k + \arctan\frac{l_{abs}}{d}\right) \end{aligned} \tag{3.7}$$

for an axisymmetric robot[3]. For a rod-like, cylindrical robot d is directly observable. To obtain a value for l_{abs} consider that all robots are driven on paths in the image plane. It can thus quite reliably be assumed that every robot will show its true absolute length at some instant in time. l_{abs} is therefore approximated at each instant t by the maximal observed length l:

$$\tilde{l}_{abs,k} = \max \mathcal{L}(k) \tag{3.8}$$

where $\mathcal{L}(k) = \{l_0, l_1, \ldots, l_k\}$ is the set of all observed robot lengths l up to and including time k.

For robots like the RodBot that are expected to move only in the image plane, θ is disregarded as well, yielding a nonholonomic kinematic state propagation law in two dimensions:

$$\mathbf{x}(k) = \begin{pmatrix} x_k \\ y_k \\ \phi_k \end{pmatrix} = \begin{pmatrix} x_{k-1} + (v_{k-1} + \frac{1}{2} * \dot{v}_{k-1}) * \cos\phi_{k-1} \\ y_{k-1} + (v_{k-1} + \frac{1}{2} * \dot{v}_{k-1}) * \sin\phi_{k-1} \\ \phi_{k-1} + \dot{\phi}_{k-1} + \frac{1}{2} * \ddot{\phi}_{k-1} \end{pmatrix} \tag{3.9}$$

Just as for unconstrained motion Equations 3.5 and 3.9 may be further simplified by assuming the velocities to be constant between time steps. To smooth the calculated results and to add robustness an EMA filter can be applied again. The Kalman filter as described in 3.1.1 however only lends itself to linear systems and an **Extended Kalman filter** has to be used instead.

In order to derive an Extended Kalman filter, the method described by R.

[3]Note that Equation 3.7 can not resolve the fourfold ambiguity between robot poses for which the robot axis encloses the same angle with the image plane. This is no issue for the given task since only the movement in the image plane is of interest.

D'Andrea [14] is used. The presented Kalman filter assumes constant velocities between time steps. The state vector $\mathbf{x}(k)$ is given by

$$\mathbf{x}(k) = \begin{pmatrix} x_k & y_k & \phi_k & \theta_k & v_k & \omega_{\phi,k} & \omega_{\theta,k} \end{pmatrix}^T \tag{3.10}$$

where $\omega_{\phi,k}$ and $\omega_{\theta,k}$ are the angular velocities of ϕ and θ at time k respectively. The measurement vector $\mathbf{z}(k)$ obtained by the visual tracking system is given by

$$\mathbf{z}(k) = \begin{pmatrix} x_{m,k} & y_{m,k} & \phi_{m,k} & l_{m,k} \end{pmatrix}^T \tag{3.11}$$

$\mathbf{x}(0)$ is initialized with the initially measured data and no velocity, its covariance matrix $\mathbf{P}_0$ with measurement noise variances where known through calculations, and estimated values indicating our confidence in the initialization data where unknown.

$$\mathbf{P}_0 = diag\begin{pmatrix} 0.005 , 0.005 , 0.0001 , 5 , 10 , 1 , 1 \end{pmatrix} \tag{3.12}$$

The following covariance matrices for process noise $\mathbf{Q}$ and measurement noise $\mathbf{R}$ follow:

$$\mathbf{Q} = diag\begin{pmatrix} 1.5 , 1.5 , 0.8 , 0.8 , 8 , 3 , 3 \end{pmatrix} \tag{3.13}$$

$$\mathbf{R} = diag\begin{pmatrix} 0.005 , 0.005 , 0.0001 , 0.005 \end{pmatrix} \tag{3.14}$$

Uncertainty is thus modeled as being mainly introduced through the velocity variables in the process models, with quite high variances since the model assumes constant velocity and accelerations therefore have to be treated as process noise. All noise is considered uncorrelated.

From Equations 3.5, 3.7 and 3.8 the state propagation law

$$\mathbf{x}(k) = \mathbf{q}_{k-1}\big(\mathbf{x}_{k-1}, \mathbf{n}_{k-1}\big) =$$

$$\begin{pmatrix} x_{k-1} + (v_{k-1} + n_{v,k-1}) * \cos(\phi_{k-1} + n_{\phi,k-1}) * \cos(\theta_{k-1} + n_{\theta,k-1}) + n_{x,k-1} \\ y_{k-1} + (v_{k-1} + n_{v,k-1}) * \sin(\phi_{k-1} + n_{\phi,k-1}) * \cos(\theta_{k-1} + n_{\theta,k-1}) + n_{y,k-1} \\ \phi_{k-1} + \omega_{\phi,k-1} + n_{\phi,k-1} + n_{\omega_\phi,k-1} \\ \theta_{k-1} + \omega_{\theta,k-1} + n_{\theta,k-1} + n_{\omega_\theta,k-1} \\ v_{k-1} + n_{v,k-1} \\ \omega_{\phi,k-1} + n_{\omega_\phi,k-1} \\ \omega_{\theta,k-1} + n_{\omega_\theta,k-1} \end{pmatrix} \tag{3.15}$$

and the measurement prediction law

$$\mathbf{z}(k) = \mathbf{h}_k\big(\mathbf{x}(k), \mathbf{w}(k)\big) = \begin{pmatrix} x_k + w_{x,k} \\ y_k + w_{y,k} \\ \phi_k + w_{\phi,k} \\ \tilde{l}_{abs,k} * \cos\theta_k + d * \sin\theta_k + w_{l,k} \end{pmatrix} \tag{3.16}$$

follow, assuming constant velocities, with $\mathbf{n}(k)$ and $\mathbf{w}(k)$ the process and measurement noise respectively. The prediction estimate $\hat{\mathbf{x}}_p(k) = \mathbf{q}_{k-1}\big(\hat{\mathbf{x}}_m(k-1), \mathbf{0}\big)$ requires no linearization but the measurement estimate $\hat{\mathbf{x}}_m(k)$ and the error propagation equations of the Kalman filter do. The time-dependent transition matrix $\mathbf{A}(k)$, measurement matrix $\mathbf{H}(k)$, noise covariance matrix $\tilde{\mathbf{Q}}(k) = \mathbf{L}(k) * \mathbf{Q} * \mathbf{L}^T(k)$ and measurement noise covariance matrix $\tilde{\mathbf{R}}(k) = \mathbf{M}(k) * \mathbf{R} * \mathbf{M}^T(k)$ are given with[4]

$$\mathbf{A}(k) = \frac{\partial \mathbf{q}_k}{\partial \mathbf{x}(k)}\big(\hat{\mathbf{x}}_m(k), 0\big) \tag{3.17}$$

$$\mathbf{H}(k) = \frac{\partial \mathbf{h}_k}{\partial \mathbf{x}(k)}\big(\hat{\mathbf{x}}_p(k), 0\big) \tag{3.18}$$

$$\mathbf{L}(k) = \frac{\partial \mathbf{q}_k}{\partial \mathbf{n}(k)}\big(\hat{\mathbf{x}}_m(k), 0\big) \tag{3.19}$$

$$\mathbf{M}(k) = \frac{\partial \mathbf{h}_k}{\partial \mathbf{w}(k)}\big(\hat{\mathbf{x}}_p(k), 0\big) \tag{3.20}$$

[4] see Appendix A.2.8

As this discussion suggests, there are multiple ways to implement prediction algorithms. Assuming that the underlying kinematic description is correct, the best choice depends on the given setting. In order to allow tracking of the objects the algorithm does not have to calculate the best possible estimate of the object state, the estimate only needs to be sufficiently close to obtain correct matches[5]. The more objects occur in the scene, the more accurate must the estimate be. But since more objects in the scene also means less time for calculating states per object, the algorithm needs to be faster as well. Accuracy and speed generally work against each other and a trade-off therefore may have to be found by the user.

A possible issue for some prediction algorithms is the direction of movement. The angle ϕ calculated using image moments does not account for the movement of the robot. The correct angle, which points in the same direction as the average velocity vector of the robot might be its opposite $\phi + \pi$. While a Kalman filter for nonholonomic motion will indicate an incorrect orientation of the angle with an estimated negative velocity, this angle has to be corrected for other filters like EMA by comparing it with the (averaged) velocity vector. Furthermore, angles have to be treated with care because of the discontinuity between two pi and zero, which is the range that was used for this work.

3.1.3 Comparison between methods

Table 2 shows a comparison of different implemented prediction algorithms for the primary model case[6]. The given data serves only as an example and gives few evidence on the best algorithm, since all the prediction results depend strongly on the parameter sets, e.g. the smoothing parameter α for exponential moving average or the $\mathbf{Q}$ and $\mathbf{R}$ matrices for Kalman filters. These have to be adjusted for different problems. For the used parameter sets, simpler algorithms like *FreeKalman*, which implements a Kalman filter for unconstrained motion, performed surprisingly well and even appear to outperform more complex algorithms like *NonHoloKalman3D*, which implements a three-dimensional Kalman filter, in the well separated case. For the aforementioned reasons it can however

[5]If interested in the best state estimates for further derivations, such can still be obtained offline by applying filtering to the raw data.

[6]Implementation details of the algorithms and plots for the distributions in the table can be found in Appendix A.2, a short summary in Section 5.1.4.

not be concluded whether this is due to the fact that the underlying equations of the more complex algorithm are ill fitted to model the problem or whether the covariance matrices chosen aren't optimal. Running further predictions with different covariance matrices would yield additional clues. While *FreeKalman* performed very well in the metrics presented in Table 2 it was the only algorithm that failed at allowing for correctly matching two robots in a situation where one closely passes by the other. This is one of two more critical situations in the primary model case where two robots are too close to each other to be recognized as two distinct objects by the algorithms described in Chapter 2. In the other critical situation one robot crosses the other. The only two state prediction algorithms which enabled successful matching of the two agents through this situation for the parameter values used in test runs were *NonHoloKalman2D* and *NonHoloKalman3D*.

Table 2: Comparison of prediction algorithm results for the start phase of the primary model case showing average predictions (pred)[ms] and update times (upd)[ms] as well as the mean error and standard deviations for the predicted distance and direction of movement and the orientation of the robot.

Algorithm	**pred**	**upd**	**distance [px]**	**direction [rad]**	**orientation [rad]**
Static	0.0014	0.005	-0.764 ± 0.7782	–	$1.7738 * 10^{-4} \pm 0.2851$
SimpleFreeMovement	0.0028	0.006	0.2862 ± 0.9336	0.0244 ± 1.1145	$54 * 10^{-4} \pm 0.3575$
FreeKalman	0.0013	0.051	-0.0716 ± 0.6745	0.006 ± 1.0609	$-1.4642 * 10^{-4} \pm 0.7831$
FreeMovingAverage	0.0020	0.010	-0.2056 ± 0.5306	-0.0773 ± 0.9516	$-18 * 10^{-4} \pm 0.2926$
NonHoloEMA	0.0013	0.014	-0.3188 ± 0.6151	-0.2240 ± 1.1306	$-22 * 10^{-4} \pm 0.3954$
NonHoloKalman2D	0.0013	0.054	-0.1666 ± 0.5909	0.2870 ± 1.0675	$337 * 10^{-4} \pm 0.1289$
NonHoloEMA3D	0.0014	0.024	-0.3188 ± 0.6151	-0.2238 ± 1.1307	$-20 * 10^{-4} \pm 0.3960$
NonHoloKalman3D	0.0013	0.063	-0.1104 ± 0.7345	0.2445 ± 1.0479	$-91 * 10^{-4} \pm 0.6751$
DirectedRodEMA	0.0020	0.016	-0.1965 ± 0.5310	-0.0727 ± 0.9500	$0.62181 * 10^{-4} \pm 0.1105$

3.2 Matching

3.2.1 Difficulties in multiple object settings

Matching in the presence of multiple objects is more challenging than for single agents and cannot be guaranteed to be performed successfully in all possible configurations. If objects are close then overshooting state predictions will lead to wrong matches, if objects touch or intersect then the extraction of correct object features with the methods presented in Chapter 2 will fail completely since all

touching objects will be considered as one connected component by the algorithm. While approaches are possible that attempt to match separate objects to locations in the combined structure, in many cases these objects will have to be considered lost and objects localized after subsequent splitting of the structure will have to be treated as new. This is especially the case if the combined structure exists for longer than just a few frames.
Another difficulty which needs to be considered is that while single objects can be approximated as convex, a combined structure of convex entities will usually have a non-convex appearance. Since the algorithms identify the position of structures in the image with their center of mass, the position of such non-convex shapes may be calculated to lie outside of the structure. If the state vector used for matching is the position in the two dimensional image space then even very close matches might prove to be false, e.g. in the case of a circular structure composed of several robots with a single robot in their middle: The object feature stemming from the latter agent might be falsely matched to the circular structure, a situation which should be prevented.

The matching procedure of the presented framework proceeds in three basic steps: In the preliminary main matching step initial matches are calculated, in a secondary step missing objects and possible matches for those are identified and in the third step a second attempt at matching is being carried out, where localization of missing objects is tackled with additional routines. The methods employed get increasingly sophisticated and time-consuming in each step as the number of objects for which the step has to be executed declines.

3.2.2 Preliminary matching

In the preliminary matching step OpenCVs brute-force matching class BFmatcher is used to match each state extracted from the current frame to a state from the set of predicted states. Brute force matching calculates a norm (the 2-norm was chosen here) for all pairs obtained by combining the query state q (from the current frame) with each state in the prediction set P and returns the predicted state m corresponding to the pair with the minimal norm:

$$m(q) = \left\{ m \middle| \left\| (m,q) \right\| = \min_{n_i \in P} \left(\left\| (n_i,q) \right\| \right), m \in P \right\} \tag{3.21}$$

For matching the state vectors are composed of the x position, the y position and the orientation angle.
In further steps the obtained matches are filtered for multiple matchings and matches exceeding a maximal matching distance are rejected. The basic principle is illustrated in Algorithm 3.

Algorithm 3 Preliminary matching

Input: set of predicted states P, set of states extracted from current frame E
Output: mapping information for predicted states, information if old objects were matched

```
for all states in E do
    find best bfmatch in P
end for
for all bfmatches do
    if matching distance too high then
        discard bfmatch
    else
        push extracted state on stack of corresponding matched predicted state
        set object corresponding to matched predicted state as being found
    end if
end for
for all stacks of predicted states do
    if stack size = 1 then
        accept mapping
    else if stack size > 1 then
        for all extracted states in stack do
            calculate distance of predicted state from contour of extracted state
        end for
        accept mapping with minimal distance
    end if
end for
```

3.2.3 Second matching step

In the second matching step the framework iterates through all missing objects calling an object class specific, more sophisticated method to find the best matching component in the current frame for each, if any. It is ignored whether the returned component was already matched to an object in the previous step. If

no match is found then the object will go missing. Groups are built for each extracted structure containing objects that calculated to be part of it. If only one object is matched to a structure the mapping is accepted.
The standard object routine to find the best matching component provided by the object type base class SceneObject is shown in Algorithm 4. It basically consists of a series of additional distance measurements.

Algorithm 4 SceneObject class: best matching component calculation

Input: sets of rotated rectangle *regions* and *contours* for all components found in the current frame
Output: best matching component *bestMatch* or nothing if no suitable match

```
matchCandidates ← {}
for all components do
    if object center inside component region then
        matchCandidates ← component
    end if
end for
if matchCandidates empty and useOldStateFallback then
    for all components do
        if old object center inside component region then
            matchCandidates ← component
        end if
    end for
end if
if matchCandidates empty then
    calculate predicted region of object
    for all components do
        calculate overlap between component contour and predicted region
    end for
    if regions with overlap found then
        matchCandidates ← component with highest overlap
    end if
end if
if matchCandidates size = 1 then
    return as bestMatch
else if matchCandidates size >1 then
    return component as bestMatch where object center closest to contour
end if
```

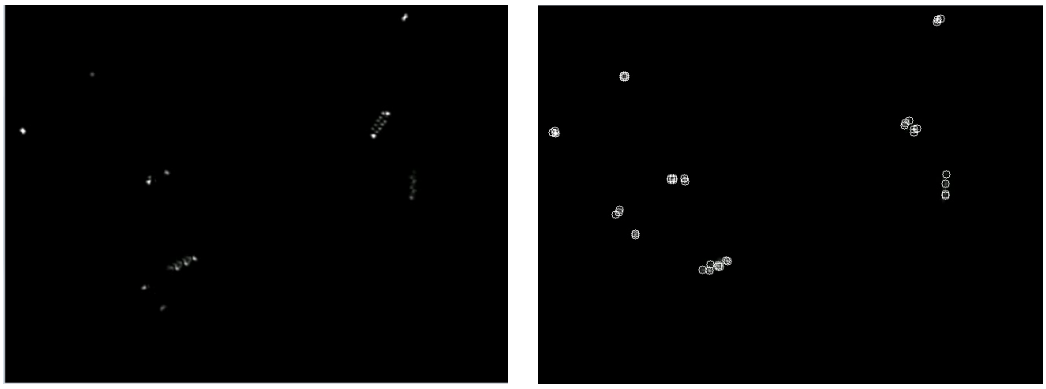
3.2.4 Missing object localization in connected components

In the final matching step localization of those objects in the groups matched to a given structure is attempted. While test runs for the tracking of objects in joint structures were carried out using different proposed techniques, e.g. optical flow, the only method that showed partial success was the use of a corner harris detector [20] to create feature sets to segment the structures. The used procedure appears to allow tracking for at least short periods of time before subsequently extracted robot positions start to loose their physical meaning. It must also be stated that the method is not very robust and a good state prediction engine is crucial for the success of tracking. The method is based on the observation that for rod like robots such as the ABFs in the primary model case the corner harris method can be tuned[7] such that it will highlight the two end points of the rod (see Figure 7). Under the assumption that the endpoints of single robots are contained in the sets obtained by applying corner harris to the joint structures, it is concluded that these points might be used as indicators for the movements of the agents inside the structure. A local maxima calculation method was therefore developed to extract the positions highlighted by the corner harris algorithm. Point pair based features are then extracted from the point clouds (x endpoint 1, y endpoint 1, x endpoint 2, y endpoint 2, corner harris intensity endpoint 1, corner harris intensity endpoint 2, edge length, edge orientation) and each dimension is weighted by experimentally determined weight factors. Skipping a few implementation details these pair features are then matched against predicted pair features and new approximated states are calculated for every object based on the matched point pair. The method would benefit from additional data to fine-tune the weighting of feature coordinates.

3.3 Object updating

If the matching procedures have finished all objects which have been successfully matched with new state sets are updated by adding their new states. Again, this is done through class specific functions using virtualization that thus may call additional program routines, e.g. to smooth the raw, new states with state filters such as moving average or Kalman filters, calculate additional states or adjust

[7] The values used are 6 for the blocksize, 5 for ksize and 0.04 for k.



(a) Corner harris detection

(b) Local maxima extraction from corner harris

Figure 7: Results of using corner harris detection to extract the endpoints of robots from the image.

the new states, e.g. by determining the direction of the orientation angle with respect to the movement of the robot. Objects that could not be matched to a new state set are set to be missing, or, if they have been missing for a minimal, user-defined period of time they are considered lost and are no longer tracked. On the other hand new state sets that were not matched to any existing object are the initial states for the instantiation of new objects.
Figures 8 and 9 show the successful tracking of the objects in the primary model case as well as in the secondary model case 1.

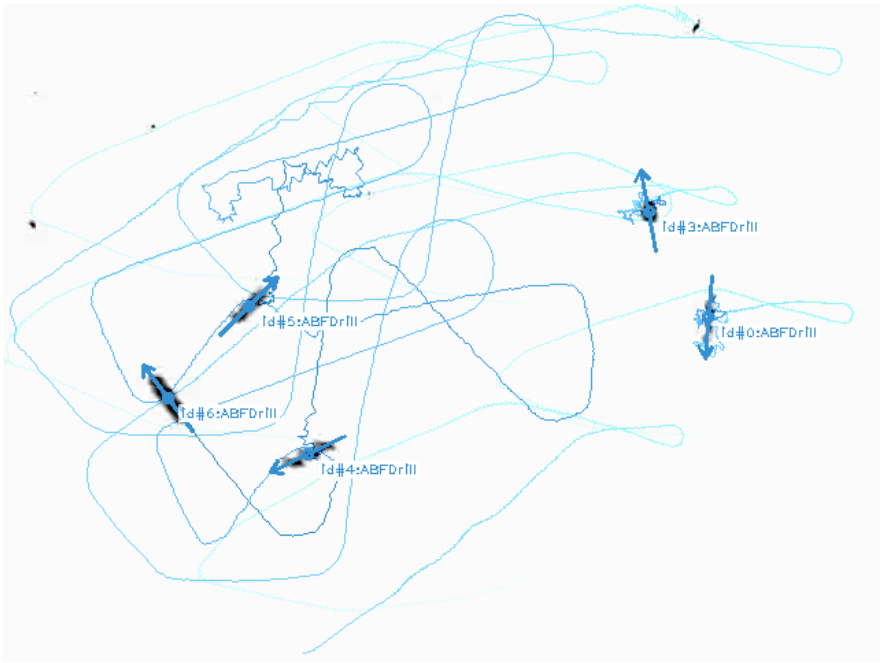


Figure 8: Tracking output example for the primary model case, displaying path(line) and orientation(arrow) of each detected object.

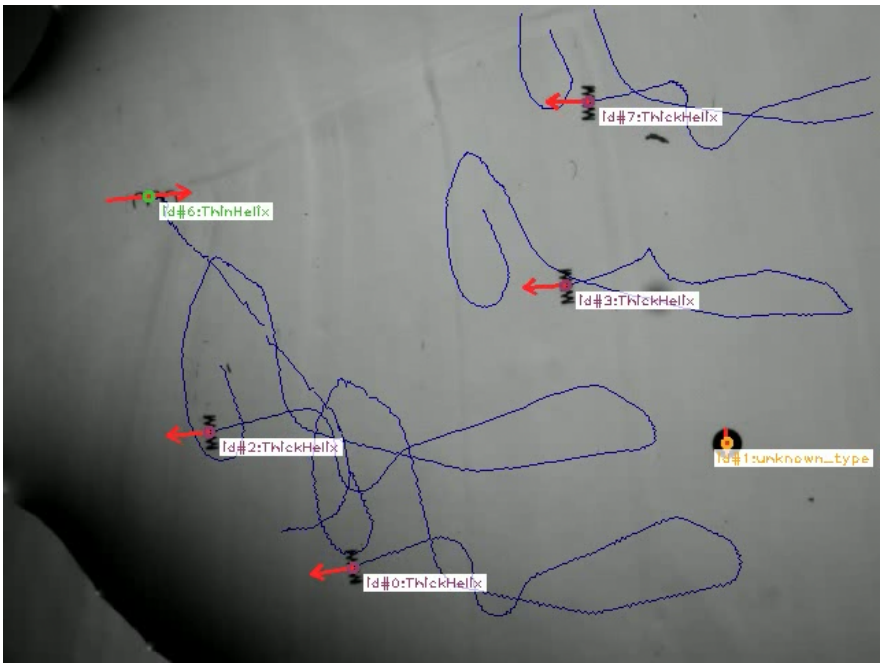


Figure 9: Tracking output example for the secondary model case 1, displaying path(line) and orientation(arrow) of each detected object.

4 Classification and type recognition

If microrobotic platforms are used to perform specified tasks they will necessarily interact with other (micro-sized) object types and in some cases more than one microrobot type might by deployed to solve an assignment. Thus, the scene observed by the camera will include two or more different object types, as seen in the secondary model cases 1 and 2. In order to handle each type correctly and exploit the type specific properties for control it is desirable that the computer is able to recognize and distinguish different kinds.

Given an algorithm for type recognition, objects are divided in two groups from a classification perspective, additionally to the three groups we introduced for matching:

- **Unclassified objects** are objects for which no classification has been found yet, either because they have only recently been detected or because there was not enough time for type distinction. As a conclusion they run standard routines only.
- **Classified objects** are objects for which the classification stage has been executed at least once. They are believed to be of a certain type and are thus running type specific routines, e.g. for state prediction.

As was already mentioned in Chapter 2.1.2 the feature descriptors extracted from scene objects appeared to be – while not useful for the calculation of object features such as position and orientation – quite promising as a classification means to distinguish object types. This idea was developed, evaluated more thoroughly and finally implemented to obtain a classification algorithm.

4.1 Using feature descriptors for classification

4.1.1 About feature descriptors

The identification of characteristic points in images and the development of descriptors for these have been studied extensively and a wide range of so called keypoint extractors and feature descriptors are available. While the implementation details of these keypoint extractors and feature descriptors vary, keypoint extractors are generally designed to efficiently localize positions in the image

where many changes can be observed (often quantified using image gradients), while feature descriptors typically calculate multidimensional vectors describing the point and its surroundings, normalized such that the features become invariant to standard image transformations such as lighting, scale and perspective changes. For these properties, feature descriptors are highly attractive in all applications dealing with the appearance of objects depicted in images.

Not all feature descriptors and keypoint extractors mentioned in the subsequent sections will be further explained, instead explanations will focus on those methods which have eventually been employed. Refer to the given references for additional information about other descriptors.

4.1.2 Feature descriptors for microrobots

For a high resolution picture of objects in the macro world, keypoint extraction and feature point descriptor routines readily produce a considerable amount of features which can then be used for further processing. Given such an amount of data, the set of features obtained from one single picture is enough to robustly locate the corresponding object in other images, as has been shown e.g. by Lowe *et al.* [23] using *scale-invariant feature transform* (SIFT) descriptors. However, test runs showed that for microrobots, e.g. the ABFs in the primary model case, a considerably lower number of feature points must be accepted, i.e. depending on the chosen extractors and descriptors between 1 and 20 feature points. Figure 10 shows obtained feature points for running a SIFT descriptor on a set of ABFs.

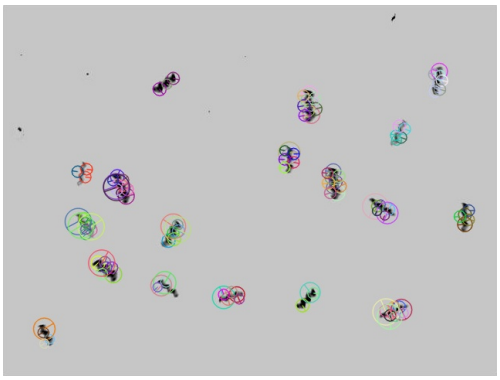


Figure 10: Scale-invariant feature transform (SIFT) feature points [23] displayed for an image with a set of ABFs.

For the combination of a *features from accelerated segment test* (FAST) point extractor [30] with a *binary robust invariant scalable keypoints* (BRISK) feature descriptor [22] an average number of 9.21 keypoints was obtained for the primary model case. Such data amounts are insufficient for robust calculations as they are too prone to errors, e.g. false matches. Therefore classification of microrobots can only be successful if considering statistical data collected over time. For this type of approach the small feature sets available per object and frame are beneficial, since this reduces the computational cost for updating the statistical data.

4.2 Class likelihood estimation

From the machine learning community algorithms, called *classifiers*, are known which are able to automatically derive decision rules based on provided training data. These decision rules segment the multidimensional data space in regions that are each assigned a specific identifier. The training data consists of samples from the data space, e.g. the feature descriptor space (typically 64- or 128-dimensional), for which the correct identifier, e.g. the robot class the feature belongs to, is known. Using the derived rules the classifier then predicts the identifier for samples whose identifier is not known. Obviously data space segmentation can only be achieved if more than one identifier is supplied to the classifier. If only one identifier is available, e.g. only one object type occurs in a setup, then classification may either be deactivated or a standard negative identifier can be provided. For this framework, the latter approach was chosen, since this adds robustness if unexpected objects appear in the scene.

A classifier as described above is the pivotal element in the proposed procedure for type recognition, which was loosely inspired by *bag of keypoints* approaches (e.g. [12]):

- **Necessary manual steps before automated object type recognition**:

    1. Assisted by the framework, the user creates sets of feature points describing a certain microobject or -robot in a given configuration. Once created, sets are stored and managed by the framework and may thus be reused in arbitrary configurations.

2. The user defines the visual appearance of specified object types by assigning a choice of previously recorded feature sets. Object types are stored and managed by the framework.
3. The user configures the framework by selecting which object types are expected to occur in a given microrobotic setting. This scene configuration can be stored and loaded.

- **Automated (offline) preparation of classification components after manual scene configuration**:

1. The feature sets corresponding to a positive decision, i.e. those describing the object type, and the ones corresponding to a negative decision, i.e. those describing another object type, are identified.
2. The framework's library of already trained classifiers is searched for the chosen feature set configuration. This provides a significant performance boost if a match is found since classifier training is computationally expensive.
3. If no match is found in the library then a new classifier is trained (using the feature set configuration as training data) and subsequently added to the library.

- **Automated (online) type recognition**:

1. A keypoint extractor and a feature point descriptor are employed to extract the feature set of the object which is to be classified.
2. For each object class its classifier predicts for each feature vector in the feature set if the vector belongs to the class.
3. For each object and each class the ratio of positively classified feature vectors to the total number of feature vectors of the object is used as an estimator for the likelihood of the object to be of the respective type. This estimation may further be transformed to an actual likelihood using an empirically derived mapping.
4. Class likelihoods are averaged over time.
5. The class with the highest likelihood is accepted as the object's type if its probability exceeds a predefined threshold, e.g. 50 %.

4.3 Evaluation of different implementations

The OpenCV library provides a wide selection of available keypoint extractors, feature descriptors and classifiers. Several combinations of these were evaluated to decide upon the best configuration for the task.
The miscellaneous classifiers mentioned in this section are handled just as the feature descriptors, i.e. only the classifier which was eventually used is being described.

4.3.1 Setup

For the lack of alternatives the primary model case was chosen as "test case" for evaluating the performance of different combinations. At the time when the classification stage was designed this was the only model case available. Since at least two object types are needed to evaluate if an algorithm configuration is able to discriminate between different types a second, artificial type class was introduced. A few examples of this set consisting of 200 hand drawn basic shapes are shown in Figure 11.

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

Figure 11: Examples from artificial type class.

In order to challenge the classification stage the shapes in the drawings were chosen such that they resembled the shapes of the ABF robots in the primary model case.

For the evaluations the artificial set was split into two sets of 100 images each, where one was used as a training set and the other as an evaluation set. Similarly a training and an evaluation set of 100 ABF robot images each were extracted from the primary model case video file.

Evaluation runs were then carried out according to the following procedure:

1. A keypoint extractor, a feature descriptor and a classifier are chosen.

2. Feature sets are created for the training and evaluation images.

3. The classifier is trained using the feature sets obtained from the training images.

4. The trained classifier is used to predict the type of the feature vectors contained in the feature sets obtained from the evaluation image sets.

5. The results of the prediction stage are quantified.

4.3.2 Results

In order to identify promising combinations first an overview was gained by calculating the overall ratios of successfully and falsely predicted feature descriptors for a large number of combinations[8]. It is noted that many feature descriptor algorithms also implement a keypoint extraction stage and can thus be used for both tasks. The results for the ABF features are shown in Table 3, those for the artificial features can be found in Appendix A.3.

Table 3: Falsely classified ABF feature vector percentages for classifier and extractor/descriptor pair combinations

classifier	**FAST/BRISK**	**SURF/SURF**	**BRISK/BRISK**	**BRISK/FREAK**
bayes	15.91%	43.16%	24.67%	58.49%
k-nearest (k=1)	13.22%	21.84%	36.67%	34.91%
k-nearest (k=3)	12.81%	23.51%	38.67%	
k-nearest (k=5)	13.64%	24.66%	43.38%	
k-nearest (k=8)	10.95%	24.56%	40.67%	
k-nearest (k=10)	12.19%	25.91%	43.33%	
svm	100%	28.21%	100%	83.02%
decision trees	25.21%	29.05%	72%	61.32%
boosting	22.31%	12.96%	25.33%	51.89%
random trees	10.43%	24.56%	45.33%	55.66%

These results indicated that either a configuration using FAST keypoint extractors and BRISK feature descriptors or one using the *Speeded-Up Robust Features* (SURF) algorithm [6] might perform best. While BRISK feature descriptors

[8]Refer to [19] for more information about the *Normal Bayes Classifier* (bayes), [4] for *K-Nearest Neighbours* (k-nearest), [11] for *Support Vector Machines* (svm), [9] for *Decision Trees*, `http://www.stat.berkeley.edu/users/breiman/RandomForests/` for *Random Trees*, [6] for *Speeded-Up Robust Features* (SURF) and [3] for *Fast Retina Keypoint* (FREAK).

performed well on the test set, BRISK keypoint extractors produced significantly lower numbers of keypoints than any other investigated extractor, affecting the performance of BRISK/BRISK and BRISK/FREAK configurations and suggesting that the features of the robots were not sufficient to pass the internal tests of the BRISK detector. The output number of keypoints for the FAST extractor on the other hand mainly depends on a configurable threshold. A few test runs showed that a threshold of 60 is a good choice.
Based on this first evaluation step, it was decided to focus further evaluation on SURF and FAST/BRISK configurations.

In the second evaluation step the ratios of descriptors classified positively as ABF robots to the total number of descriptors were calculated for all images in the evaluation sets and the results were investigated using histograms, showing the type distinction ability of the different configurations. The configuration which performed best was using a boosting classifier along with SURF descriptors. It is depicted in Figure 12. Refer to Appendix A.3 for the results of other classifiers.

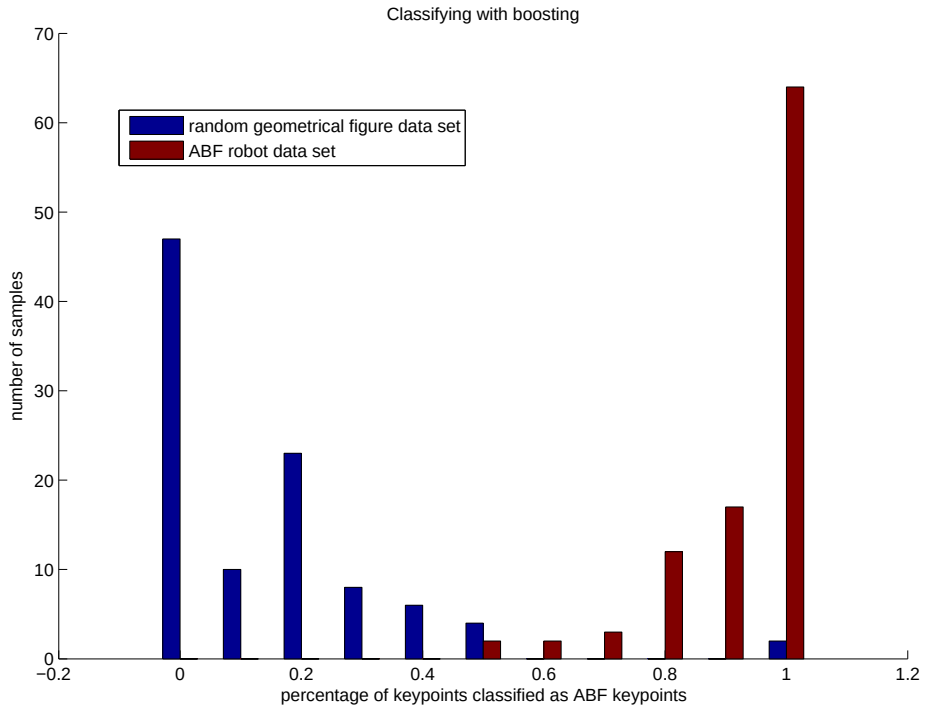


Figure 12: Histogram result using a boosting classifier and SURF feature descriptors.

Ratio values are plotted against the x-axis, the number of images identified with the respective value against the y-axis. Results for the artificial type are colored blue, those for the ABF robot type red. Observe that the proposed estimator using the ratio of positively classified feature descriptors is indeed a very good measure for type distinction in this configuration, e.g. even a primitive classification function using a threshold of 50 % creates no false negatives for the ABF evaluation set while only classifying about 8 % of the artificial evaluation set falsely. Using an empirically derived mapping, e.g. based on ratios of robots classified correctly to robots classified falsely for any given ratio, and using averaging of likelihoods over time, object type distinction and thus classification are improved further.
Chronometries show that SURF requires an average computation time of about 4 ms per object. Real-time processing with this constraint is still feasible even for multiple objects since not every object in a scene has to be classified at each instant of time, but this will slow down classification for larger sets of objects. The FAST extraction routine on the other hand finishes in 0.11 ms, the BRISK description routine in 0.28 ms on average. Since this is a considerable difference, the less optimal though still sufficient distinction abilities of the FAST/BRISK approaches were chosen for speed gain. As boosting and k-nearest neighbours classifiers performed comparably for FAST/BRISK, the decision to choose boosting was based on the timing characteristics of the classifiers, shown in Table 4.

Table 4: Experimentally determined prediction times of classifiers

classifier	**average time used for prediction step**
Normal bayes classifier	0.11 ms
K-nearest neighbours	1.3 ms
Decision trees	0.05 ms
Boosting	0.1 ms
Random trees	0.25 ms

Figure 13 shows the histogram obtained for the evaluation case when using boosting with FAST/BRISK features. The results of other classifiers with FAST/BRISK can be found in Appendix A.3.

Features from accelerated segment test (FAST) [30] is a corner detection algorithm tailored at high throughput. It detects corner points in an image by ex-

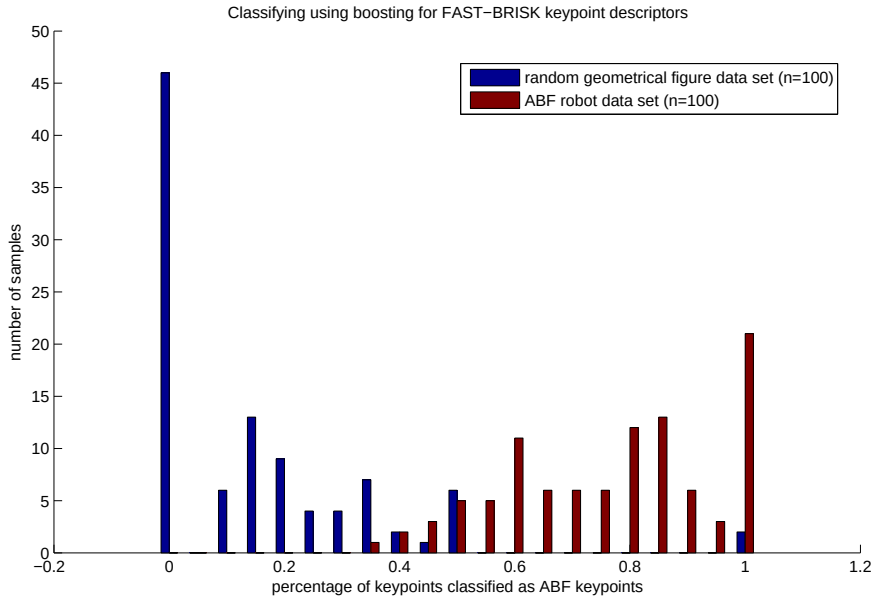


Figure 13: Histogram result using a boosting classifier, a FAST keypoint extractor and BRISK feature descriptors.

amining pixels on a circle surrounding the corner candidate. If a certain number of contiguous pixels is brighter than the candidate plus a configurable threshold and another number is darker than the candidate minus the threshold, then the candidate is classified as corner and added to the set of returned points.
The *binary robust invariant scalable keypoints* (BRISK) [22] method calculates a bit-string descriptor of length 512 (thus 64-dimensional vectors using 8-bit integers) for every keypoint, assembled via gaussian smoothed brightness comparisons of points on a rotation- and scale-normalized circular sampling pattern around the keypoint which is being described.
Finally *boosting* [18] is a method that combines the performance of many weak classifiers[9] to build a strong classifier. During the training process, weak classifiers are repeatedly trained on the data set, the training errors are calculated and the weights of misclassified samples are increased and then normalized. The final classifier is the sign of the weighted sum over the individual weak classifiers. In OpenCV decision trees are used as weak classifiers which decide about the output value by applying a threshold. Test runs showed that *boosting* performs better

[9] A weak classifier is only required to perform better than chance and may thus be designed using a simple and computationally inexpensive algorithm.

if the feature set sizes for the types occuring in the scene are of about the same order (and thus have the same weight in the set), which is why the framework approximately equals all set sizes in case of notable differences by duplicating the smaller ones.

4.4 Verification for model cases

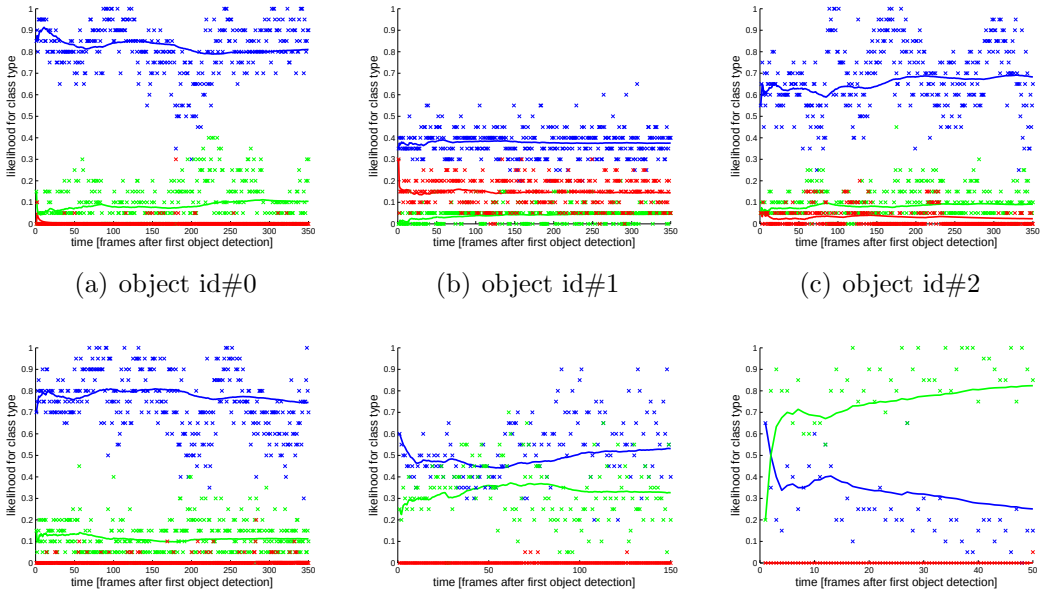
Successful classification and type recognition in real-time was shown for all three model cases (using the artificial set as a negative for the primary model case). Since only the two secondary model cases feature more than one type, only these are presented in this section.

4.4.1 Secondary model case 1

Three video files were available for the two microrobot types occuring in the secondary model case 1. Two of these files were used to generate the feature descriptor sets while successful classification was shown on the third. The two types in the model case are easily discriminable for the human eye, one shows a thick helix moving perpendicular to its axis, the other a thin helix with its direction of motion aligned to its axis. For the thicker helix type the two video files yielded a feature set with 36,135 descriptors, for the thinner type one with 10,036 descriptors. In such a case the algorithm automatically quadruplicates the smaller feature set, which resulted in a new size of 40,144 descriptors for the thinner type. In order to fragment the feature space further, the artificial type set was added as well, yielding additional 19*1902 = 36,138 features. This enabled the classification stage to successfully identify a third unknown type occuring in the scene (see Figure 15). The total size of features with which the three classifiers for the three types (ThickHelix, ThinHelix, Artificial) were trained was thus 112,417. The required computing time necessary for training each classifier was about 140 s.
A linear mapping function was used to map ratios to likelihoods: Ratios were thus accepted directly as likelihood estimator. For recognition as a certain type a minimal likelihood of 50 % was required.

Results are visible in Figures 14 and 15. For robots 0, 2 and 3 type recognition as a thick helix was carried out safely from the beginning while the classification

stage struggled more with robot 4, for which correct classification took more time (about 100 frames). The thin helix robot 6 was falsely classified for the first few frames but then safely recognized. The unknown object 1 was successfully identified as an outlier with the addition of the artificial feature set, as already mentioned. Without this addition it was classified falsely as type thick helix. The data shows that even after this supplementary addition the likelihood for the thick helix type is still estimated quite high with almost 40 %. Still these results suggest that an additional more relevant negative data set, e.g. for the environment, might be beneficial and allow detection of unknown objects.
The complete plots with all classification results from secondary model case 1 can be found in Appendix A.4.1.



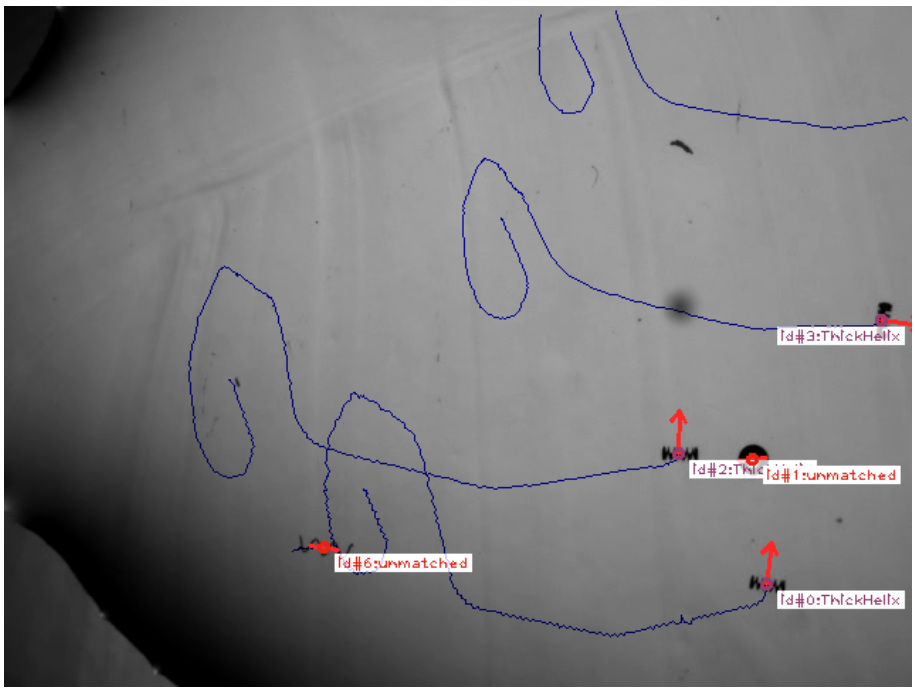
(a) object id#0 (b) object id#1 (c) object id#2

(d) object id#3 (e) object id#4 (f) object id#6

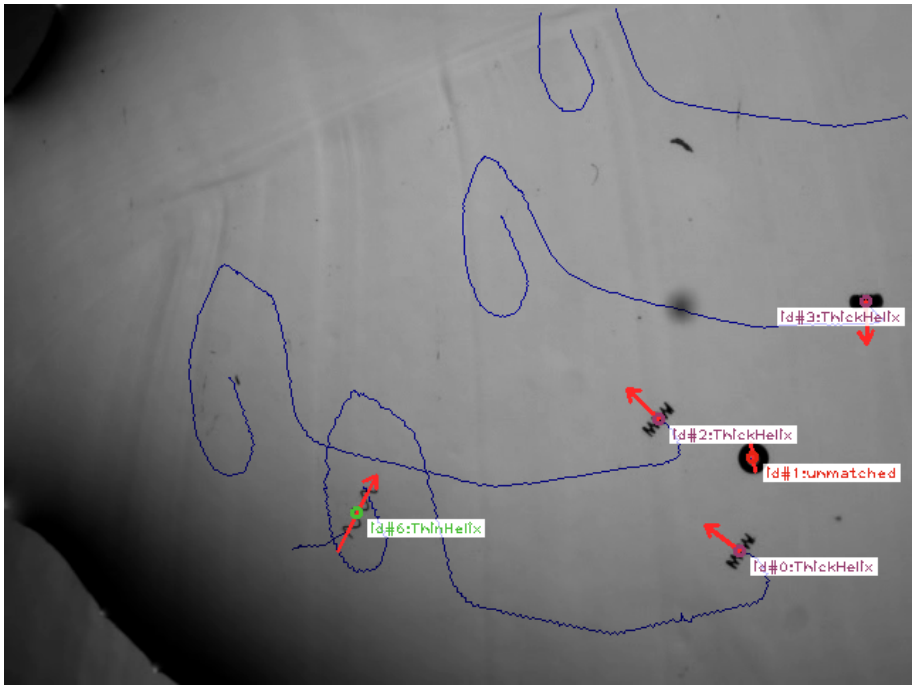
Figure 14: Type likelihood estimations for objects in secondary model case 1: Both the measurements (x) and the calculated means (line) of estimated likelihoods based on the positively classified feature ratio are shown for the *thick helix* (blue), the *thin helix* (green) and the *artificial* (red) type.

4.4.2 Secondary model case 2

Classification of a scene containing crystals was chosen as a more challenging case with a promising and direct application as discussed in the introduction. Using



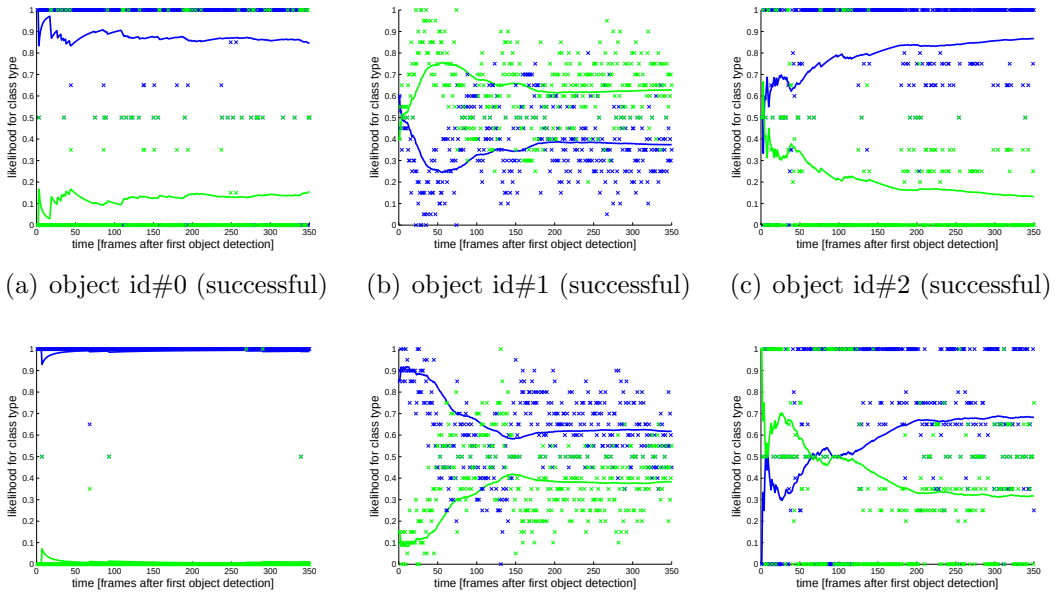
(a) t = 9 s



(b) t = 10 s

Figure 15: Secondary model 2: At t = 9 s a new microrobot (id#6) enters in a scene already containing four microobjects of two distinct types and is at first classified as an unknown type (unmatched). With the new imagery obtained during the next second, the robot is successfully classified at t = 10 s as type *ThinHelix*.

two video files to obtain training data as for secondary model case 1, classification prediction was attempted in a third. However, successful distinction between the two types failed. Using only these three files it is not possible to argue whether this was due to a high dissimilarity between crystals or due to insufficient feature set sizes. The second explanation seems likely though, since the two training files combined yielded only 4,967 features for the RodBot and 7,673 features for crystals. In order to get any features for the crystals, a simple dynamic adaption of the threshold used for the FAST keypoint extractor was introduced, as a value of 60 proved too high. The method simply decrements the threshold as long as no features are found for an object.
Successful distinction was only possible after adding features obtained from the second half of the classifying video to the training set and then predict the type of the objects in the first half. 958 additional crystal- and 2,866 additional RodBot-features were obtained. The results are shown in Figure 16 a)-d).



(a) object id#0 (successful) (b) object id#1 (successful) (c) object id#2 (successful)

(d) object id#4 (successful) (e) object id#1 (unsuccessful) (f) object id#2 (successful)

Figure 16: Type likelihood estimations for objects in secondary model case 2 with (a-d) and without (e,f) additional features: Both the measurements (x) and the calculated means (line) of estimated likelihoods based on the positively classified feature ratio are shown for the *RodBot* (green) and the *crystal* (blue) type.

Even with the additional features it is clearly visible how classification of the

crystals with the id's 0 and 4 succeeds without any, classification of crystal 2 with few problems. The prediction results of the RodBot with id 1 on the other hand are wider spread and thus less certain. In Figure 16 e) and f) classification results for the RodBot and the crystal with id 2 are shown without the additional features. While classification of the RodBot fails completely, classification of the crystal succeeds after an initialization time of about 75 frames. The complete data sets using additional features can be found in Appendix A.4.2, the ones without in Appendix A.4.3. Figure 17 shows successful classification and tracking in the secondary model case 2 with the additional features.

More experiments using more data will be necessary in order to validate if the proposed technique may be used to distinguish microrobots and crystals or not.

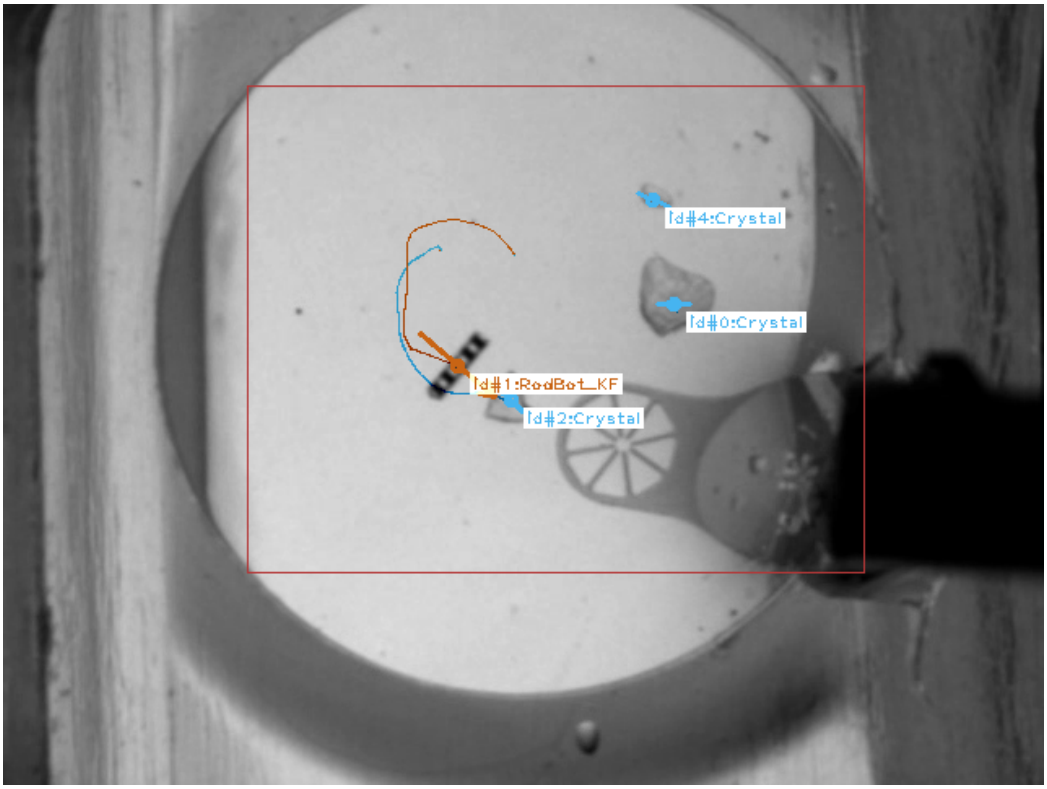


Figure 17: Tracking and successful classification of crystals and a RodBot in the case with additional features. The red rectangle indicates the observed part of the image stream.

5 The MOLAR framework

All the algorithms and functionality described in the previous chapters have been integrated in a framework which will subsequently be called MOLAR, an acronym for *multiple object localization and recognition*. It was designed to give easy access to all functionality while being easily extendable through the use of factories. Furthermore featured are a graphical user interface (GUI) and extensible markup language (xml) file functionality, which are used to save and load settings for types, features and the framework. MOLAR also saves the current configuration if the program is closed.
This chapter gives an insight into and overview of certain features and functionality of the framework.

5.1 Object class system and generic types

It was already mentioned in Chapter 3 that each object detected in the imagery is being instantiated as its own data object. Thus, instead of having a data vector with entities of objects in the current and previous frames, e.g. the position, on which class objects operate, each object governs its own data and is instantiated as the type which it is currently believed to be. Wherever possible data operations are delegated to the object. This has the advantage that through virtualization functionality of objects can be fitted to object type specific requirements, e.g. an object recognized as a free-floating crystal can execute a different dynamic model than a microrobot with known non-holonomic constraints.

5.1.1 Parent class

To achieve this, each hard-coded object type inherits from the *SceneObject* class, which defines all necessary interface functionality used by other modules of the framework. The *SceneObject* class is not a pure-virtual, thus abstract class and may be instantiated. In fact, every object is instantiated as a SceneObject at first, indicating that its type is unknown. Currently virtualized functionality includes state prediction and addition (a derived class may thus infer more object properties than the state vector provided by the framework, see Chapter 3), appearance (drawing) on the output image, update of class likelihoods and class update, object information and data export, and the second step in the

matching pipeline. The SceneObject class furthermore provides the factory for all types implemented, where the user may register additional types through a simple registration function, and an object identification system through unique numbering. Since the SceneObject class is the parent class for all object types it also serves as a type conversion bridge if the type, which an object is believed to be, changes. This is necessary since the object type is not known at instantiation and may furthermore change indeterminant times if classification is inconclusive (see Chapter 4). Thus on type change a new object is instantiated from the old object and the old object is destroyed. No data loss occurs for states, i.e. if the previous type calculated additional information this data isn't lost but possibly not accessible through the new type until type conversion back to the old type.

5.1.2 Abstraction and generic type

In order to simplify the creation of new object types an abstract type structure was derived with which any object class may be described. It is composed of three elements:

- The **header** contains all informational data of the object class, e.g. the class name, a description, class color and the creation date.
- The **classification module** describes the appearance of the object type in images and is used for classification.
- The **dynamics module** holds, filters, predicts, draws and exports the object's state.

Using these three building blocks a generic class was implemented with which the user can create new types at runtime by combining existing modules. While the header module is simply specified by the user using text forms, the classification and dynamics module are explained further in the following sections.

5.1.3 Classification module

As was discussed in Chapter 4 a *boosting* classifier is used to classify objects based on *BRISK* feature descriptors obtained from image data. In order to allow generic class creation, these steps were separated. Feature descriptor sets can be created at runtime, described and stored to the hard disk. To define a new

class, MOLAR lists all known feature descriptor sets and the user may choose an arbitrary combination of those to describe the appearance of the new object type.
When a new stream is being opened the user tells the framework which object types are expected to occur in the scene and the framework automatically trains a classifier for each class, based on the feature set combination of the objects. MOLAR also stores trained classifiers and recognizes if a classifier has already been trained on a specific feature combination before, loading the old results to avoid lengthy recalculations.

5.1.4 Dynamics module

While different types may have a different appearance, their dynamics are often governed by the same equations. Different types can thus be divided in different groups with the same or a similar dynamic behavior, e.g. holonomic and non-holonomic robots. The framework currently depends on the knowledge of the user to choose the appropriate module from the list of available dynamics. Since different dynamics usually implies different code, new dynamic modules can not be created at runtime but have to be coded, compiled and registered with the dynamics factory. However, the framework also includes an interface for parameters, which means that a specific module may be further individualized by overwriting its standard parameters, e.g. the filter length for moving average filters or the state transition matrix for a Kalman filter. All these parameters are accessible through the .xml-configuration files, there is no recompilation involved. New dynamic modules may be defined by deriving from the *Dynamics* class, which also serves as factory.

The algorithms previously presented in Table 2 are the modules that have been implemented so far and may be chosen for new object types:

- **Static** models a static object that does not change its state.
- **SimpleFreeMovement** describes unconstrained motion with three degrees of freedom (position x and y, orientation). Prediction is based on a constant acceleration assumption. No state filters are applied.
- **FreeKalman** implements a Kalman filter for unconstrained motion with

three degrees of freedom (position x and y, orientation) while assuming constant velocities.

- **FreeMovingAverage** describes unconstrained motion with three degrees of freedom as well. All values are filtered using an exponential moving average filter for position, velocity and acceleration separately. These are also used for prediction.
- **NonHoloEMA** describes non-holonomic motion in two dimensions, filtering the position with an exponential moving average filter.
- **NonHoloKalman2D** implements a Kalman filter for non-holonomic motion in two dimensions, assuming constant velocities.
- **NonHoloEMA3D** describes non-holonomic motion in three dimensions and filters the states using exponential moving average filters.
- **NonHoloKalman3D** implements a Kalman filter for non-holonomic motion in three dimensions, using the model described in Section 3.1.2.
- **DirectedRodEMA** applies exponential moving average filtering for unconstrained movements in a 4-dimensional parameter space, targeted at rod-like objects and predicting orientation changes resulting from rotations in planes orthogonal to the image plane by observing the object's length.

For the non-holonomic motion algorithms in two dimensions additional modules for objects moving orthogonally to their axis are available.

5.2 The main routine

The main routine subsequently executes the steps described in Chapters 2, 3 and 4 for each image passed to the framework. Following a few implementation details are given.

5.2.1 Image filtering

New image filters may be introduced by deriving the *IPAlgorithm* class and registering the new classes with the image algorithm factory, which is provided by *IPAlgorithm* as well. MOLAR offers two stages where image filtering may be

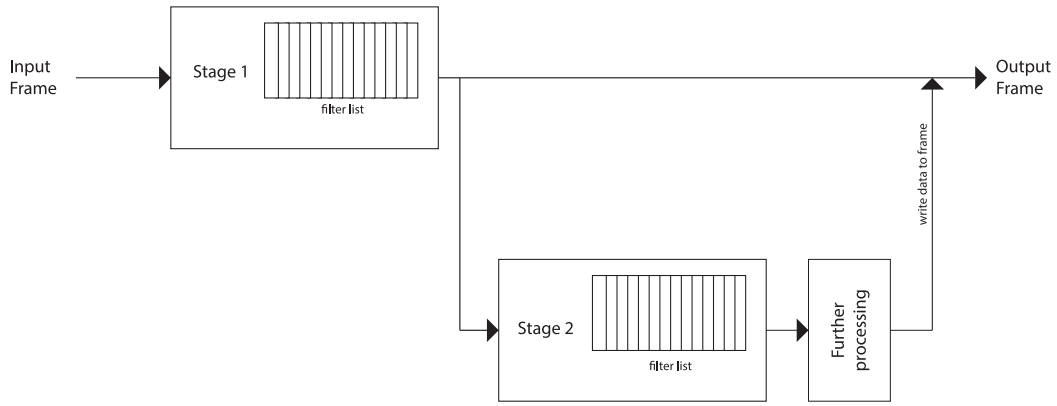


Figure 18: Image filtering pipeline showing the image data flow in the framework.

performed, as is shown in Figure 18. Image filters applied in stage 1 do not only affect the internal imagery but also the main output which is displayed to the user, while stage 2 is the last preparation step before thresholding and does not affect the visible main output. Both stages may hold an arbitrary number of filters, which are applied in the order specified. The filters, their order and their parameters can be changed at runtime and each filter type may be added more than once to any stage.

5.2.2 Classification

Classification is performed using two first-in-first-out (FIFO) queues. Unclassified objects are pushed on the priority queue while classified objects are pushed on the standard queue. Using timing aware classification the framework monitors the time used for classifying objects and estimates if the time left for the current frame is sufficient to perform another classification. If that is the case, then the next object to be classified is taken from the priority queue until the queue is empty. Afterwards objects are retrieved from the standard queue. It is thus ensured that all objects will get classified and reclassified subsequently even if not enough time is given to classify all objects at each time step. An optional time overhead can be set to enforce a margin between the time used and hard time limits. If timing aware classification is deactivated, the classification stage will be run once for all objects, neglecting any timeouts.

5.3 Chronometries

The time measurements for the three model cases on an Intel Core i7 CPU (K875 @ 2.93 GHz) processor listed in Table 5 show that the framework is perfectly able to handle real time processing. The available time overhead allows the framework to counterbalance time peaks while leaving enough room for further processing. The targeted time overhead set for the three runs was 5 ms.

Table 5: Chronometries for the three model cases.

Measurement	**Primary**	**secondary 1**	**secondary 2**
Total time [ms]	20.2401 ± 2.5914	19.5905 ± 2.0105	13.5056 ± 3.1122
Max time [ms]	59	53	30
Min time [ms]	15	13	8
Time violations [nr]	25	13	0
Time violations [%]	0.4162	0.4589	0
Mean violation [ms]	5.8400 ± 5.9279	3.8462 ± 5.2257	-
Preprocessing [ms]	4.8645 ± 0.5790	0.5930 ± 0.6192	0.4832 ± 0.5870
Various [ms]	3.7331 ± 0.5576	3.7907 ± 0.5788	3.8783 ± 0.4599
Internal prepr. [ms]	0.0027 ± 0.0515	1.461 ± 0.6113	0.5739 ± 0.4948
Localization [ms]	3.2570 ± 0.8074	3.3720 ± 0.9113	2.1516 ± 0.8877
Matching [ms]	0.7836 ± 0.8377	0.9788 ± 2.0105	1.7863 ± 2.8133

For the primary model case the *color range expansion* algorithm was used in the preprocessing stage and the *NonHoloKalman2D* dynamics module for handling the object states.

For the secondary model case 1 the *contrast brightness adjustment* algorithm was used for internal preprocessing and the *NonHoloKalman2D* dynamics module and its orthogonal counterpart for handling the object states.

The secondary model case 2 was filtered using the *contrast brightness adjustment* algorithm for preprocessing. To store, smooth and predict the states the orthogonal *NonHoloKalman2D* dynamics module was employed for the RodBot and the *FreeMovingAverage* dynamics module for the crystals.

The *various* time measurement includes the time used for tasks such as frame buffering, cropping the observed area and extracting a single channel greyscale image for further processing. Included in the *total time* measurement are besides the tasks listed the classification stage and drawing paths and objects into the output image. The chronometries in Table 5 were obtained for the framework running with graphical user interface (GUI). Without the GUI, even better times

can be obtained where only the first iteration (the initialization run) violates the 30 fps constraint.

5.4 Graphical user interface

A program can never be better than its interface. It is therefore of outermost importance that the interface gives convenient access to all important features, is well-arranged and speeds up the expected workflow instead of interfering with it. Optimally it should be also intuitively understandable for users who were not engaged in its development.

The graphical user interface of MOLAR was implemented using *Qt Meta Language* (QML), which is a user interface markup language to develop interfaces with Qt[2]. Its main window is shown in Figure 19. On the left hand side but-

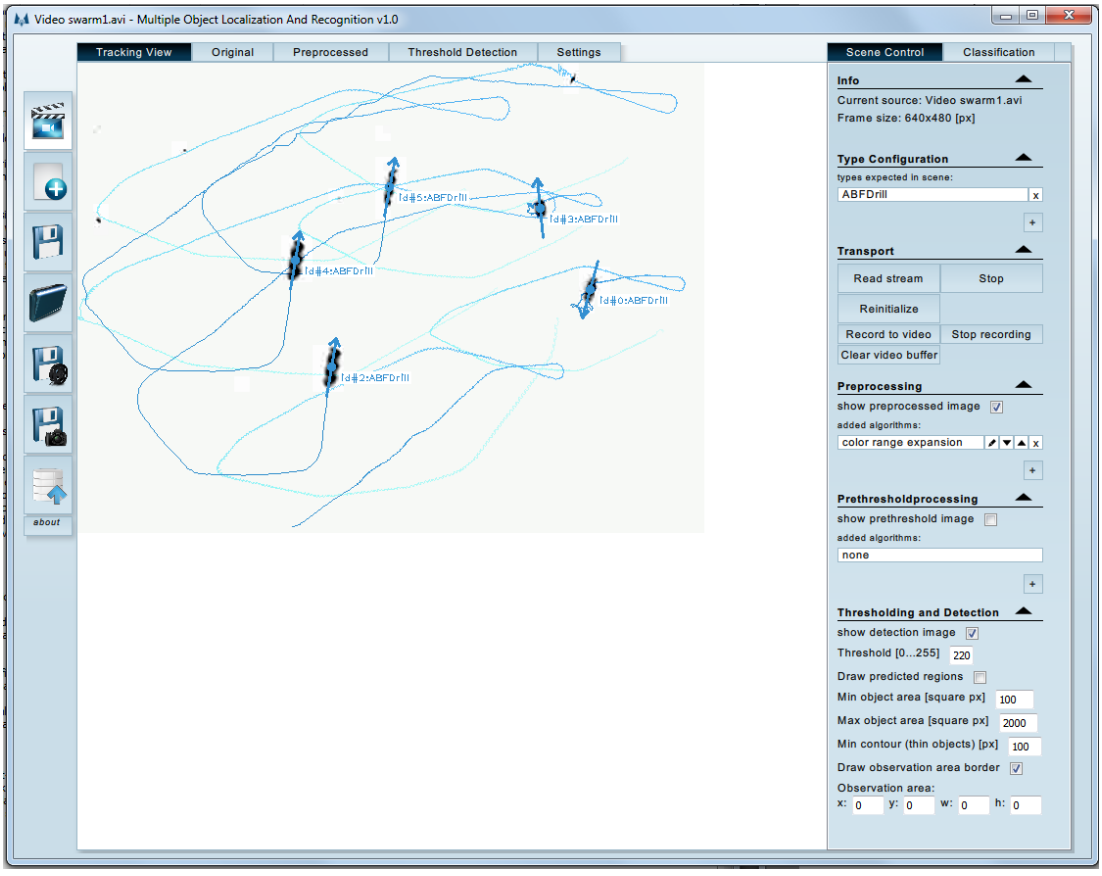


Figure 19: Main window of MOLAR's graphical user interface.

tons to load, save, import and export project configurations, streams and data

(to text, video or image files) are located. The middle section is used to display stream data and general program configurations, which can be chosen using the tabs located at the top. Additionally to the *Tracking view* showing the detected objects and their paths (depending on the configuration) and the *Original* view, which shows the original input stream, the images created in the two image filtering and the threshold stages may be displayed as well in order to ease parameter tuning. Finally the section on the right hand side is used to control the current scene, giving access to type, image filter, threshold and detection configurations. Choosing the *Classification* tab new feature sets and object types can be defined here as well.

6 Outlook

Several microrobots have been and will be developed at the Institute of Robotics and Intelligent Systems (IRIS) of ETH Zurich, two of which – the ABF and RodBot robots – were featured in this work. The presented versatility of the framework allows it to yield good results in various settings and will therefore be especially beneficial when new types are introduced and new experiments and setups are designed. MOLAR hopefully will prove to be useful in very different situations and for an extended period of time. Still, this will require continued code maintenance, updates and improvements, matching it to new requirements. It was designed to lend itself easily to such and is commented well. A choice of the very next possible future steps along with suggested improvements are listed in the subsections of this chapter.

6.1 Next steps and future work

- **Integration with MSRL systems:** In a very next step the framework will have to be integrated with the existing systems and software (such as the Daedalus framework) at the Multi-Scale Robotics Lab (MSRL) in order to investigate its performance further in multiple situations and to pinpoint weaknesses and possible improvements.

- **Integration with ROS:** The Robot Operating System (ROS) is currently establishing itself as the reference for robot control software libraries. It thus would make sense to integrate MOLAR with ROS functionalities.

- **Experiments:** The framework features many parameters for which the determination of best values would benefit from further data. Besides, machine learning techniques could be applied to dynamically change and update configurations.

- **Integration of control:** Type distinction will show its full potential if the control loop is closed and the framework's data is used to control and steer multiple objects in the scene. A new, fourth control module can be added to the generic type class, which will enable type dependent control.

- **Interobject communication:** If objects are controlled through the framework, their instantiation as actual objects in the program will allow for an implementation of some sort of communication between different robots and the control unit on a code level, e.g. allowing them to make requests for inputs to the centralized control engine, which might enable interesting control schemes.

- **Crystal harvesting:** More experiments and data are needed to validate if crystals can be safely recognized. These are also scenes that are more challenging for the thresholding stage and thus additional, specialized filters might be necessary. If these problems are overcome however, then the framework – integrated with a control unit – will be very useful for the automated harvesting of crystals.

6.2 Suggested improvements

- **Performance:** Processes could be sped up by using OpenCV's functionality to use graphic card processors. Additionally the option to use region of interest fragmentation could be given to the user in order to gain time in cases where the total number of objects occuring in the scene is fix throughout operation. Furthermore, storing even more intermediate results than already implemented might improve speed.

- **Memory load:** Possibly partly due to the usage of QML for GUI programming the main memory load is currently very high. This should be further investigated and the GUI should be reprogrammed according to the results. It seems likely that QML excessively loads libraries that might be unnecessary.

- **Debugging:** The program as it is has already been debugged partly. It still remains a first version though and the existence of further problems and so called *bugs* appears likely.

- **Refactorization:** The program code would benefit from refactoring as it has grown quite large during the project (about 15,000 lines of code, including hard-coded object types). Data and program flow as well as

object handling and class structure probably could be improved knowing the final structure of the framework.

- **GUI:** The graphical user interface does not yet offer access to all features of the framework. Some parameters as those of dynamic modules are currently only accessible through the respective XML files. These could be included in the GUI as well.

- **Statistics** A statistics module which evaluates the quality of the prediction stages and monitors execution times would be interesting.

- **Classification** The classification stage could be extended. Other classifier types could be dynamically chosen, additional object and classification properties could be incorporated. If prediction results of type classes are evaluated and the available time allows for multiple state predictions, then classes with a worse prediction than other classes should be marked as being less likely.

- **Object crossings:** New and better algorithms are needed to track objects over crossings and intersections. Those may be implemented in a type-dependent way. It might thus make sense to virtualize the third matching stage and delegate the task to the objects themselves.

- **Exploiting of swarm-like behavior:** Groups of the same robot type often move simultaneously. This could be exploited to detect and track objects past image borders.

7 Summary

A framework for real-time localization and recognition of multiple microsized objects in an image stream has been presented. The proposed methods have been shown to succeed at the given tasks and especially real-time type classification combined with virtualization of object functionality is a promising approach that can be extended further for future applications. Through the use of factories for image filters and object types, the implementation of a generic type with a factory for its dynamic modules and the use of xml files to save and load configurations of the framework, of filters and objects, the framework is versatile and easily extendable. Wherever possible, algorithms and program design were kept in a general way, limiting the number of requirements for image stream and contained robot and object types that are given by the framework itself. Instead, requirements for the stream depend on the used image filters while type dependent routines were mostly delegated to the objects themselves, utilizing class virtualization. The framework is thus also equipped well for further adjustments.
For artificial bacterial flagella several combinations of feature point detectors and classifiers were tested for their classification performance and finally FAST detectors combined with BRISK descriptors and OpenCV's Boost classifier were accepted as the best solution. It was shown that the combination is able to distinct between different helix type microrobots as well and that it yields promising results for the distinction between RodBots and crystals, but that further experiments will be needed for verification.
For prediction and state estimation a set of possible algorithm choices available with the framework was presented, consisting of exponential moving average and Kalman filters.
In a next step multiple object control algorithms should be developed and implemented, using the data provided by this framework to steer groups of robots and to perform tasks involving different microobject types.

References

[1] Opencv library documentation. `http://docs.opencv.org`, September 2014.

[2] Qt project. `http://qt-project.org/`, September 2014.

[3] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast Retina Keypoint. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 510–517. IEEE, June 2012.

[4] N. S. Altman. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*, 46(3):175–185, Aug. 1992.

[5] M. A. Andrade, F. Buiochi, and J. C. Adamowski. Particle manipulation by ultrasonic progressive waves. *Physics Procedia*, 3(1):283–288, 2010.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[7] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin. Reconfiguring massive particle swarms with limited, global control. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8243 LNCS, pages 51–66, 2013.

[8] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. 2008.

[9] L. Breiman, J. Friedman, C. Stone, and R. Olshen. Classification and regression trees. 1984.

[10] R. G. Brown. Exponential Smoothing for predicting demand. *OPERATIONS RESEARCH*, 1957.

[11] C. J. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, June 1998.

[12] G. Csurka and C. Dance. Visual categorization with bags of keypoints. *Workshop on statistical ...*, 2004.

[13] J. E. Curtis, B. A. Koss, and D. G. Grier. Dynamic holographic optical tweezers. *Optics Communications*, 207:169–175, 2002.

[14] R. D'Andrea. Recursive Estimation. Printed Lecture Notes. ETH Zurich, 2014.

[15] E. Diller, S. Floyd, C. Pawashe, and M. Sitti. Control of Multiple Heterogeneous Magnetic Microrobots in Two Dimensions on Nonspecialized Surfaces. *IEEE Transactions on Robotics*, 28:172–182, 2012.

[16] B. R. Donald, C. G. Levey, I. Paprotny, and D. Rus. Simultaneous Control of Multiple MEMS Microrobots. In G. S. Chirikjian, H. Choset, M. Morales, and T. Murphey, editors, *Springer Tracts in Advanced Robotics Vol. 57*, volume 57 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, 2009.

[17] J. Drenth. *Principles of Protein X-Ray Crystallography*, volume 0. 2007.

[18] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337–407, Apr. 2000.

[19] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. 1990.

[20] C. Harris and M. Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, pages 147–151, 1988.

[21] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, pages 35–45, 1960.

[22] S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*, pages 2548–2555. IEEE, Nov. 2011.

[23] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004.

[24] S. Martel and M. Mohammadi. Using a swarm of self-propelled natural microrobots in the form of flagellated bacteria to perform complex micro-

assembly tasks. In *2010 IEEE International Conference on Robotics and Automation*, pages 500–505, 2010.

[25] B. J. Nelson, I. K. Kaliakatsos, and J. J. Abbott. Microrobots for minimally invasive medicine. *Annual review of biomedical engineering*, 12:55–85, Aug. 2010.

[26] K. E. Peyer, L. Zhang, and B. J. Nelson. Localized non-contact manipulation using artificial bacterial flagella. *Applied Physics Letters*, 99(17):174101, Oct. 2011.

[27] K. E. Peyer, L. Zhang, and B. J. Nelson. Bio-inspired magnetic swimming microrobots for biomedical applications. *Nanoscale*, 5(4):1259–72, Feb. 2013.

[28] R. Pieters, H.-W. Tung, D. F. Sargent, and B. J. Nelson. Non-contact Manipulation for Automated Protein Crystal Harvesting using a Rolling Microrobot. In *Preprints of the 19th World Congress of the International Federation of Automatic Control*, 2014.

[29] F. Qiu, R. Mhanna, L. Zhang, Y. Ding, S. Fujita, and B. J. Nelson. Artificial bacterial flagella functionalized with temperature-sensitive liposomes for controlled release. *Sensors and Actuators B: Chemical*, 196:676–681, June 2014.

[30] E. Rosten and T. Drummond. Machine Learning for High-Speed Corner Detection. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision ECCV 2006*, pages 430–443. Springer Berlin Heidelberg, 2006.

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

[32] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[33] K. Takahashi, N. Ogawa, H. Oku, and K. Hashimoto. Organized motion control of a lot of microorganisms using visual feedback. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 1408–1413.

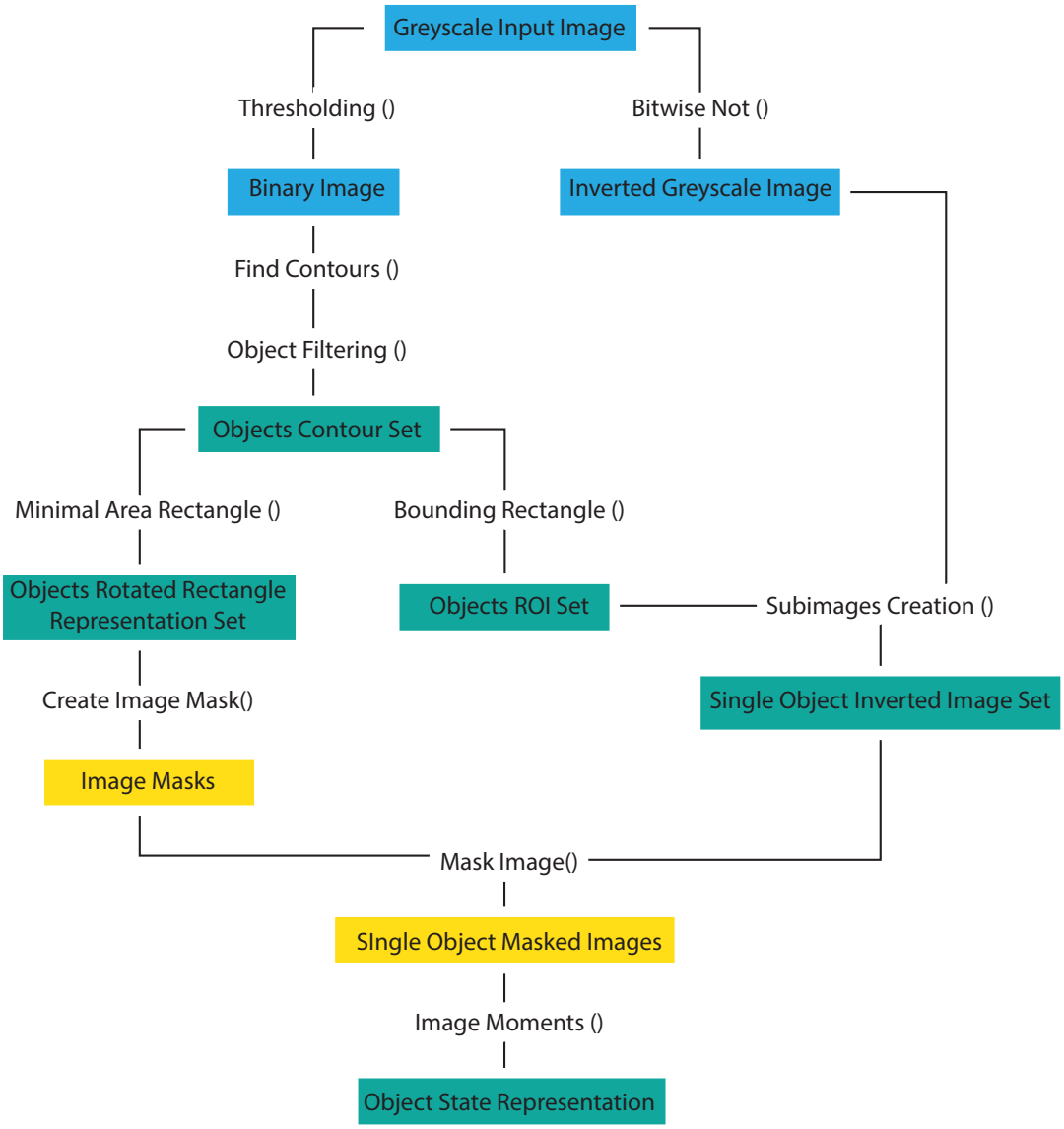
[34] H.-W. Tung, D. F. Sargent, and B. J. Nelson. Protein crystal harvesting using the RodBot: a wireless mobile microrobot. *Journal of Applied Crystallography*, 47(2):692–700, Mar. 2014.

[35] Wikipedia. X-ray crystallography. `http:/http://en.wikipedia.org/wiki/X-ray_crystallography`, September 2014.

[36] L. Zhang, K. E. Peyer, and B. J. Nelson. Artificial bacterial flagella for micromanipulation. *Lab on a chip*, 10:2203–2215, 2010.

A Appendix

A.1 Object localization data flow diagram

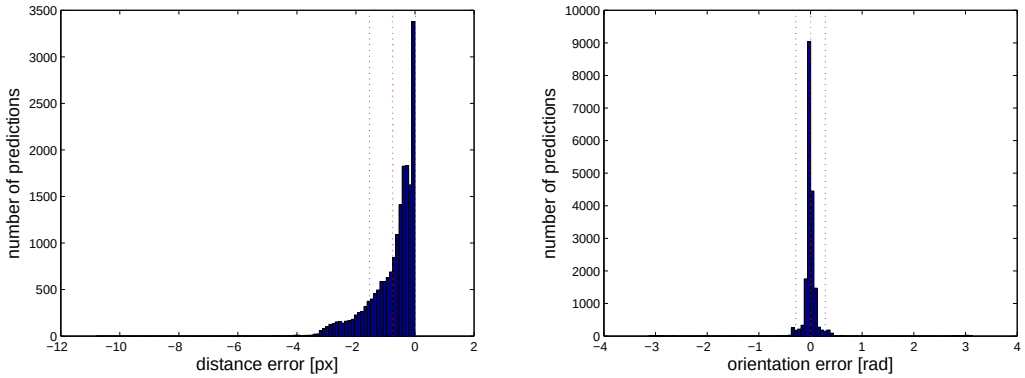


A.2 Prediction algorithms

Measured values in the equations are marked with a "m" subscript, e.g. x_m, predicted with a "p" subscript, e.g. x_p. Estimated values are indicated with a hat, e.g. $\hat{x}$.

A.2.1 Static

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



Prediction:

$$x_p = x_{k-1}$$

$$y_p = y_{k-1}$$

$$\phi_p = \phi_{k-1}$$

Update:

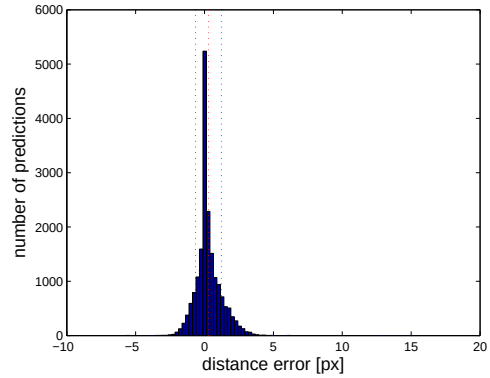
$$x_k = x_m$$

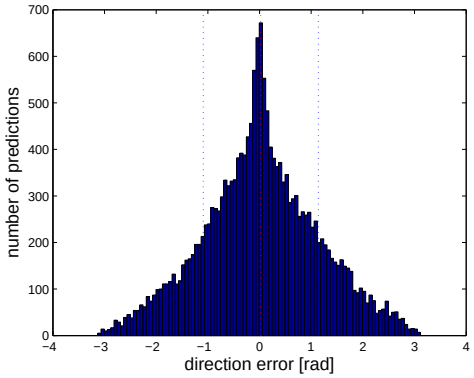
$$y_k = y_m$$

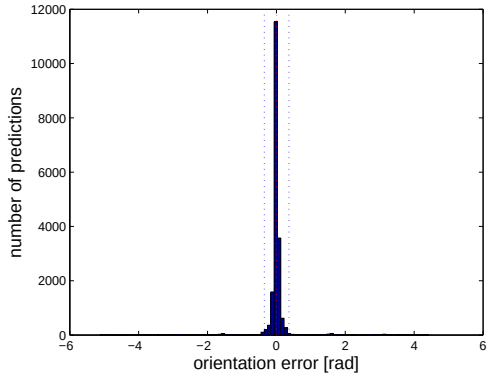
$$\phi_k = \phi_m$$

A.2.2 SimpleFreeMovement

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):







Prediction:

$$x_p = x_{k-1} + \dot{x}_{k-1} + 0.5 * \ddot{x}_{k-1}$$

$$y_p = y_{k-1} + \dot{y}_{k-1} + 0.5 * \ddot{y}_{k-1}$$

$$\phi_p = \phi_{k-1} + \dot{\phi}_{k-1} + 0.5 * \ddot{\phi}_{k-1}$$

Update:

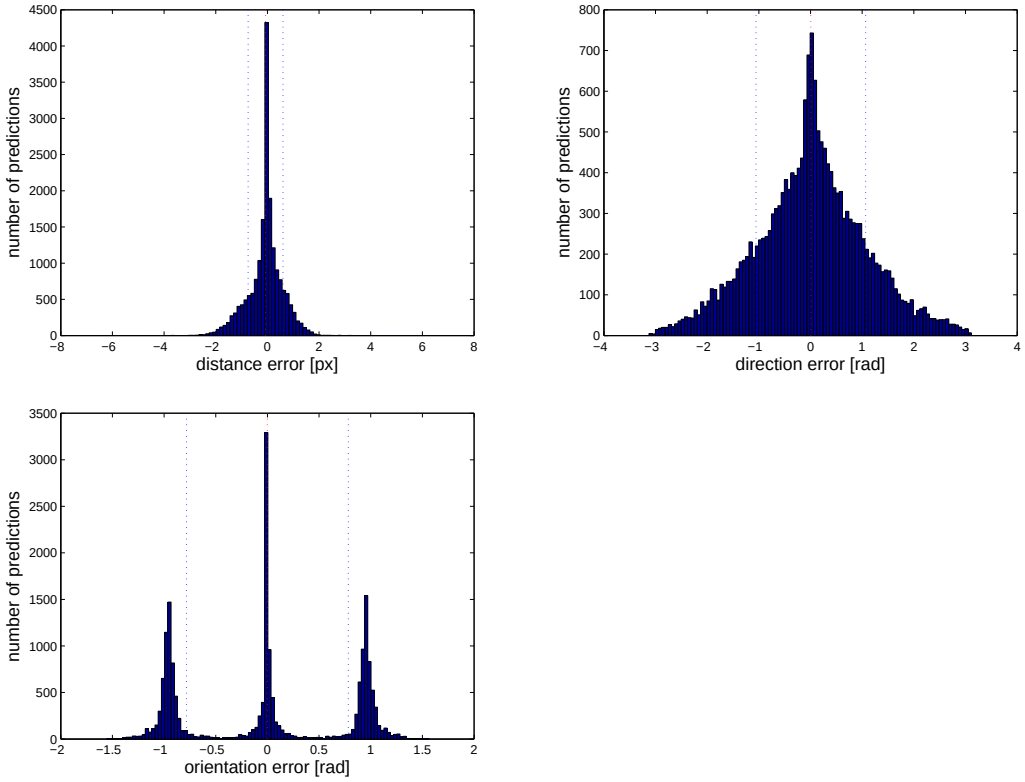
$$x_k = x_m$$

$$y_k = y_m$$

$$\phi_k = \phi_m$$

A.2.3 FreeKalman

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



State updates and predictions are carried out according to the standard Kalman equations [21]. The state vector is

$$\mathbf{x}(k) = \begin{pmatrix} x_k & y_k & \phi_k & \dot{x}_k & \dot{y}_k & \dot{\phi}_k \end{pmatrix}^T$$

The measurement vector is given by

$$\mathbf{x}_m(k) = \begin{pmatrix} x_m & y_m & \phi_m \end{pmatrix}^T$$

Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$.

The transition matrix $\mathbf{A}$ and the measurement matrix $\mathbf{H}$ used are

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix},$$

The process noise covariance matrix $\mathbf{Q}$ and the measurement noise covariance matrix $\mathbf{R}$ used are

$$\mathbf{Q} = \begin{pmatrix} 1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

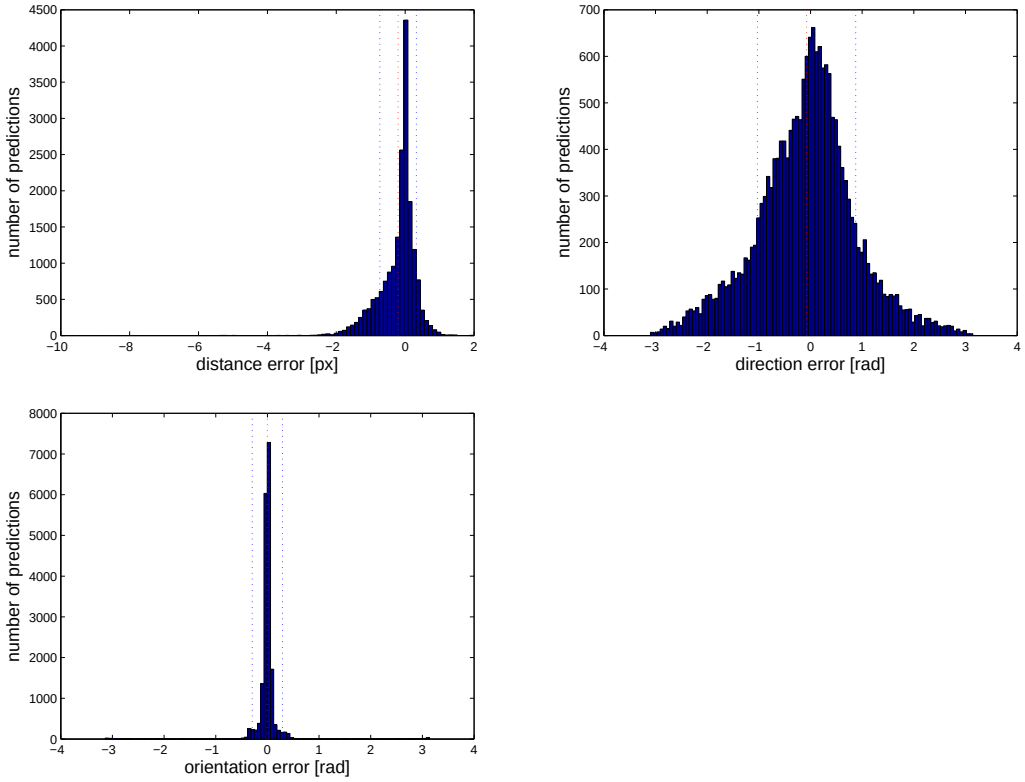
$$\mathbf{R} = \begin{pmatrix} 0.005 & 0 & 0 \\ 0 & 0.005 & 0 \\ 0 & 0 & 0.0001 \end{pmatrix},$$

The posterior error covariance matrix $\mathbf{P}_m$ is initialized with

$$\mathbf{P}_m = \begin{pmatrix} 0.005 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.005 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

A.2.4 FreeMovingAverage

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



Prediction:

$$x_p = \hat{x}_{k-1} + \hat{v}_{x,k-1} + 0.5 * \hat{a}_{x,k-1}$$

$$y_p = \hat{y}_{k-1} + \hat{v}_{y,k-1} + 0.5 * \hat{a}_{y,k-1}$$

$$\phi_p = \hat{\phi}_{k-1} + \hat{v}_{\phi,k-1} + 0.5 * \hat{a}_{\phi,k-1}$$

Update:
Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$.

$$\hat{x}_k = \alpha_{pos} * x_m + (1 - \alpha_{pos}) * \hat{x}_{k-1}$$

$$\hat{y}_k = \alpha_{pos} * y_m + (1 - \alpha_{pos}) * \hat{y}_{k-1}$$

$$\hat{\phi}_k = \alpha_\phi * \phi_m + (1 - \alpha_\phi) * \hat{\phi}_{k-1}$$

$$\hat{v}_{x,k} = \alpha_{vel} * \dot{x}_m + (1 - \alpha_{vel}) * \hat{v}_{x,k-1}$$

$$\hat{v}_{y,k} = \alpha_{vel} * \dot{y}_m + (1 - \alpha_{vel}) * \hat{v}_{y,k-1}$$

$$\hat{\phi}_{y,k} = \alpha_{\dot{\phi}} * \dot{y}_m + (1 - \alpha_{\dot{\phi}}) * \hat{\phi}_{y,k-1}$$

$$\hat{a}_{x,k} = \alpha_{acc} * \ddot{x}_m + (1 - \alpha_{acc}) * \hat{a}_{x,k-1}$$

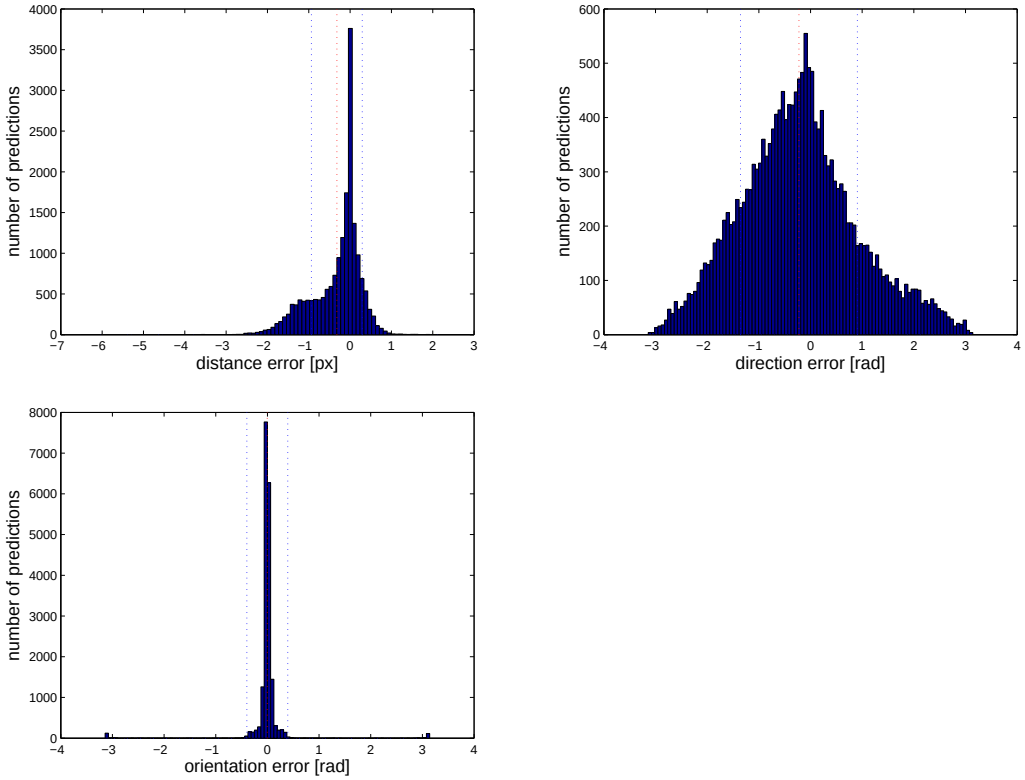
$$\hat{a}_{y,k} = \alpha_{acc} * \ddot{y}_m + (1 - \alpha_{acc}) * \hat{a}_{y,k-1}$$

$$\hat{a}_{\phi,k} = \alpha_{\ddot{\phi}} * \ddot{\phi}_m + (1 - \alpha_{\ddot{\phi}}) * \hat{a}_{\phi,k-1}$$

Standard parameters:
$\alpha_{pos} = 1, \alpha_\phi = 1, \alpha_{vel} = 0.08, \alpha_{\dot{\phi}} = 0.08, \alpha_{acc} = 0.08, \alpha_{\ddot{\phi}} = 0.08$

A.2.5 NonHoloEMA

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



Prediction:

$$x_p = \hat{x}_{k-1} + \hat{v}_{k-1} * \cos\hat{\phi}_{k-1}$$

$$y_p = \hat{y}_{k-1} + \hat{v}_{k-1} * \sin\hat{\phi}_{k-1}$$

$$\phi_p = \hat{\phi}_{k-1} + \dot{\hat{\phi}}_{k-1}$$

Update:
Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$.

$$\hat{x}_k = \alpha_{pos} * x_m + (1 - \alpha_{pos}) * \hat{x}_{k-1}$$

$$\hat{y}_k = \alpha_{pos} * y_m + (1 - \alpha_{pos}) * \hat{y}_{k-1}$$

Orientation:

$$\hat{\phi}_k = \begin{cases} \alpha_\phi * \phi_m + (1 - \alpha_\phi) * \hat{\phi}_{k-1} & , s_k \geq 0.5 \\ \alpha_\phi * \phi_m + (1 - \alpha_\phi) * \hat{\phi}_{k-1} + \pi & , s_k < 0.5 \end{cases}$$

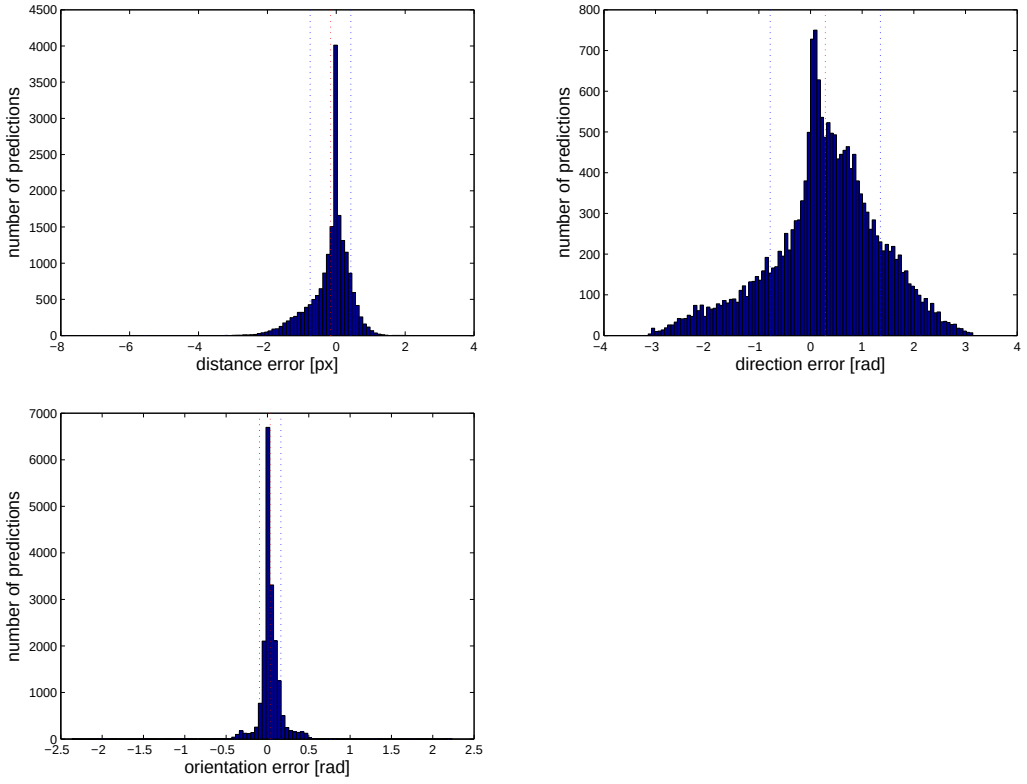
$$s_k = \alpha_s * s_m + (1 - \alpha_s) * s_{k-1} \quad ; s_0 = 0.5$$

$$s_m = \begin{cases} 1 & , \cos\phi_m * \hat{x}_k + \sin\phi_m * \hat{y}_k \geq 0 \\ 0 & , \cos\phi_m * \hat{x}_k + \sin\phi_m * \hat{y}_k < 0 \end{cases}$$

Standard parameters:
$\alpha_{pos} = 0.4, \alpha_\phi = 0.8, \alpha_s = 0.05$

A.2.6 NonHoloKalman2D

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



State updates and predictions are carried out according to the method described by D'Andrea [14]. The state vector is

$$\mathbf{x}(k) = \begin{pmatrix} x_k & y_k & \phi_k & v_k & \dot{\phi}_k \end{pmatrix}^T$$

The measurement vector is given by

$$\mathbf{x}_m(k) = \begin{pmatrix} x_m & y_m & \phi_m \end{pmatrix}^T$$

Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$.
The measurement matrix $\mathbf{H}$, the process noise covariance matrix $\mathbf{Q}$, the measurement noise covariance matrix $\mathbf{R}$ and the measurement noise projection matrix $\mathbf{M}$

are constant:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{Q} = \begin{pmatrix} 1.5 & 0 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 & 0 \\ 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0.6 \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The transition matrix $\mathbf{A}$ and the process noise projection matrix $\mathbf{L}$ are time-varying and need to be updated for every calculation:

$$\mathbf{A}(k) = \begin{pmatrix} 1 & 0 & -v_{k-1} * \sin\phi_{k-1} & \cos\phi_{k-1} & 0 \\ 0 & 1 & v_{k-1} * \cos\phi_{k-1} & \sin\phi_{k-1} & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

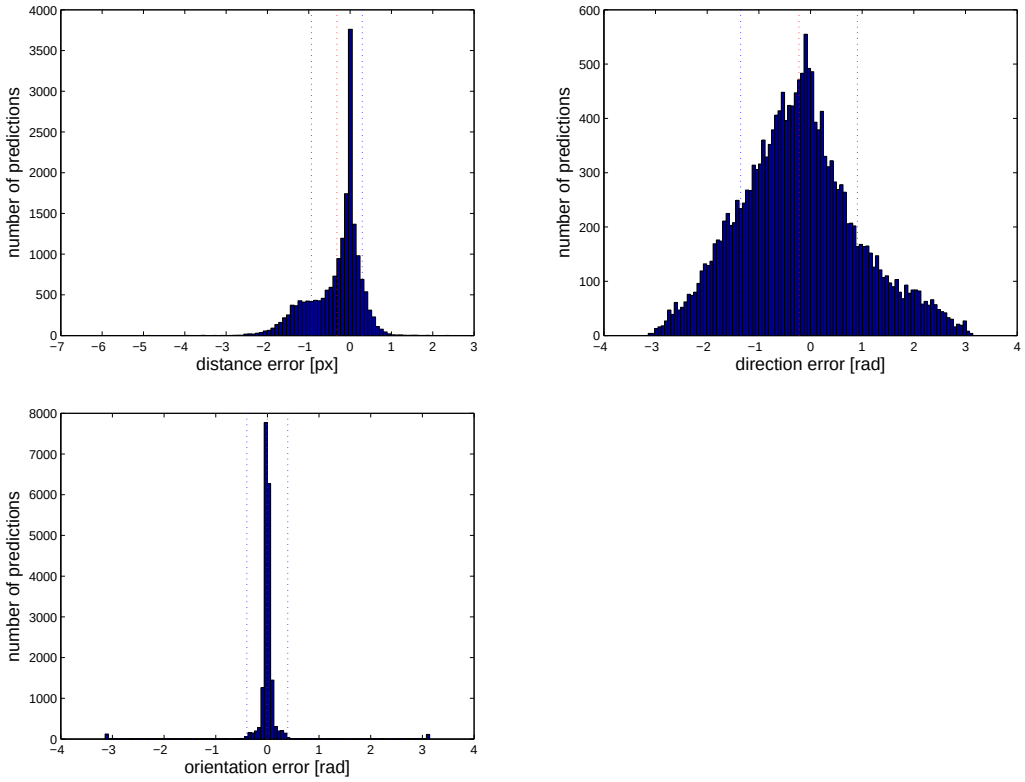
$$\mathbf{L}(k) = \begin{pmatrix} 1 & 0 & -v_{k-1} * \sin\phi_{k-1} & \cos\phi_{k-1} & 0 \\ 0 & 1 & v_{k-1} * \cos\phi_{k-1} & \sin\phi_{k-1} & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The posterior error covariance matrix $\mathbf{P}_m$ is initialized with

$$\mathbf{P}_m = \begin{pmatrix} 0.005 & 0 & 0 & 0 & 0 \\ 0 & 0.005 & 0 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

A.2.7 NonHoloEMA3D

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



Prediction:

$$x_p = \hat{x}_{k-1} + \hat{v}_{k-1} * \cos\hat{\phi}_{k-1} * \cos\hat{\theta}_{k-1}$$

$$y_p = \hat{y}_{k-1} + \hat{v}_{k-1} * \sin\hat{\phi}_{k-1} * \cos\hat{\theta}_{k-1}$$

$$\phi_p = \begin{cases} \hat{\phi}_{k-1} + \dot{\hat{\phi}}_{k-1} & , \theta_p \leq \frac{\pi}{2} \\ \hat{\phi}_{k-1} + \dot{\hat{\phi}}_{k-1} + \pi & , \theta_p > \frac{\pi}{2} \end{cases}$$

$$\theta_p = \hat{\theta}_{k-1} + \dot{\hat{\theta}}_{k-1}$$

Update:
Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$. θ_m is estimated using the measured current length l_k and the current estimates for robot length $\tilde{l}_{abs,k}$ and diameter d as described in Section 3.1.2.

$$\theta_m = \arcsin\left(\frac{l_m}{\sqrt{\tilde{l}_{abs,k}^2 + d^2}}\right) - \arctan\left(\frac{\tilde{l}_{abs,k}}{d}\right)$$

Before θ_m is used, it is compared to previous computations, just as ϕ_m.

Position update:

$$\hat{x}_k = \alpha_{pos} * x_m + (1 - \alpha_{pos}) * \hat{x}_{k-1}$$

$$\hat{y}_k = \alpha_{pos} * y_m + (1 - \alpha_{pos}) * \hat{y}_{k-1}$$

Orientation:

$$\hat{\theta}_k = \alpha_\phi * \theta_m + (1 - \alpha_\phi) * \hat{\theta}_{k-1}$$

$$\hat{\phi}_k = \begin{cases} \alpha_\phi * \phi_m + (1 - \alpha_\phi) * \hat{\phi}_{k-1} & , s_k \geq 0.5 \quad \wedge \quad \theta_p \leq \frac{\pi}{2} \\ \alpha_\phi * \phi_m + (1 - \alpha_\phi) * \hat{\phi}_{k-1} + \pi & , s_k < 0.5 \quad \vee \quad \theta_p > \frac{\pi}{2} \end{cases}$$

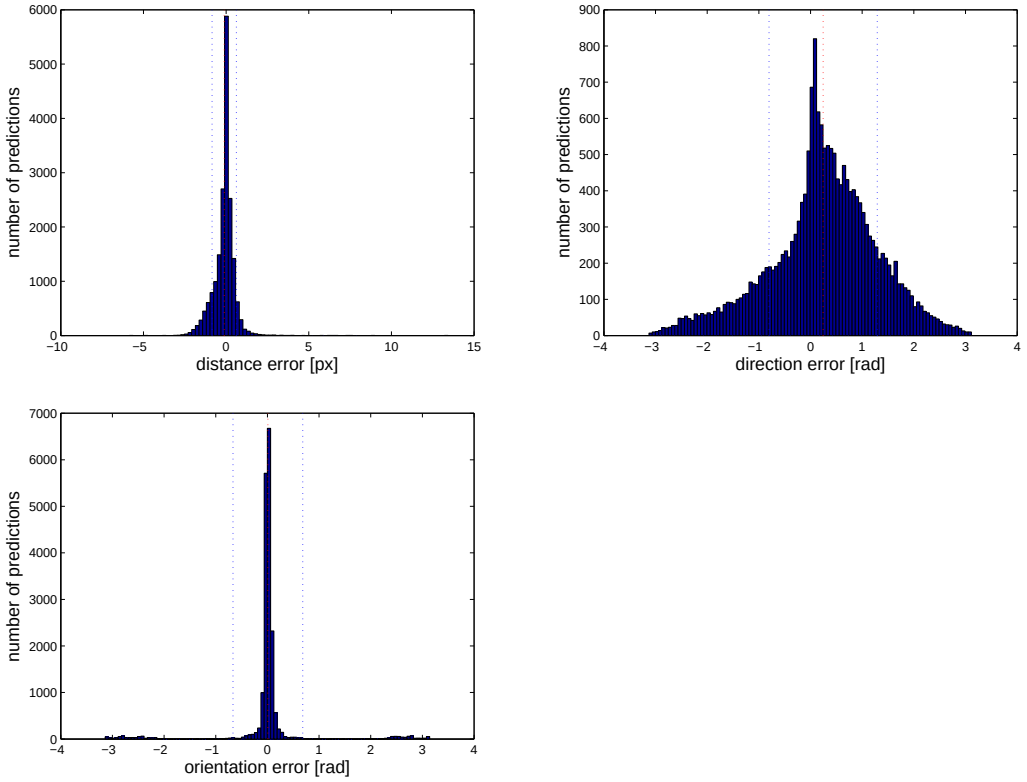
$$s_k = \alpha_s * s_m + (1 - \alpha_s) * s_{k-1} \quad ; s_0 = 0.5$$

$$s_m = \begin{cases} 1 & , \cos\phi_m * \hat{x}_k + \sin\phi_m * \hat{y}_k \geq 0 \\ 0 & , \cos\phi_m * \hat{x}_k + \sin\phi_m * \hat{y}_k < 0 \end{cases}$$

Standard parameters:
$\alpha_{pos} = 0.4, \alpha_\phi = 0.8, \alpha_s = 0.05$

A.2.8 NonHoloKalman3D

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):



State updates and predictions are carried out according to the method presented by D'Andrea [14] as described in Section 3.1.2. The state vector is

$$\mathbf{x}(k) = \begin{pmatrix} x_k & y_k & \phi_k & \theta_k & v_k & \dot{\phi}_k & \dot{\theta}_k \end{pmatrix}^T$$

The measurement vector is given by

$$\mathbf{x}_m(k) = \begin{pmatrix} x_m & y_m & \phi_m & l_k \end{pmatrix}^T$$

Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$.
The process noise covariance matrix $\mathbf{Q}$, the measurement noise covariance matrix

$\mathbf{R}$ and the measurement noise projection matrix $\mathbf{M}$ are constant:

$$\mathbf{Q} = \begin{pmatrix} 1.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.35 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 9 \end{pmatrix}$$

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The transition matrix $\mathbf{A}$, the measurement matrix $\mathbf{H}$ and the process noise projection matrix $\mathbf{L}$ are time-dependent and need to be updated for every calculation. The estimates for robot length $\tilde{l}_{abs,k}$ and diameter d are estimated separately as described in Section 3.1.2.

$$\mathbf{A}(k) = \begin{pmatrix} 1 & 0 & -v_{k-1}\sin\phi_{k-1}\cos\theta_{k-1} & -v_{k-1}\cos\phi_{k-1}\sin\theta_{k-1} & \cos\phi_{k-1}\cos\theta_{k-1} & 0 & 0 \\ 0 & 1 & v_{k-1}\cos\phi_{k-1}\cos\theta_{k-1} & -v_{k-1}\sin\phi_{k-1}\sin\theta_{k-1} & \sin\phi_{k-1}\cos\theta_{k-1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{H}(k) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\tilde{l}_{abs,k} * \sin\theta_{k-1} + d * \cos\theta_{k-1} & 0 & 0 & 0 \end{pmatrix}$$

$\mathbf{L}(k) =$

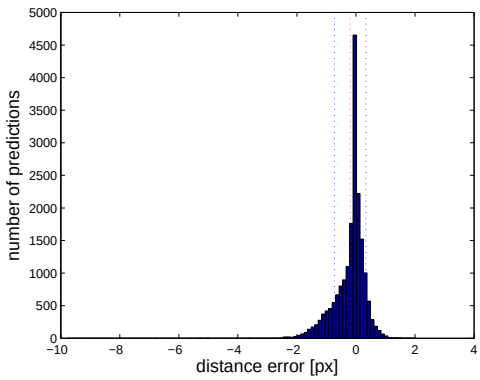
$$\begin{pmatrix} 1 & 0 & -v_{k-1}\sin\phi_{k-1}\cos\theta_{k-1} & -v_{k-1}\cos\phi_{k-1}\sin\theta_{k-1} & \cos\phi_{k-1}\cos\theta_{k-1} & 0 & 0 \\ 0 & 1 & v_{k-1}\cos\phi_{k-1}\cos\theta_{k-1} & -v_{k-1}\sin\phi_{k-1}\sin\theta_{k-1} & \sin\phi_{k-1}\cos\theta_{k-1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

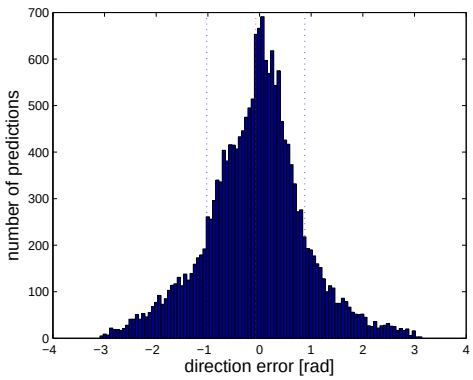
The posterior error covariance matrix $\mathbf{P}_m$ is initialized with

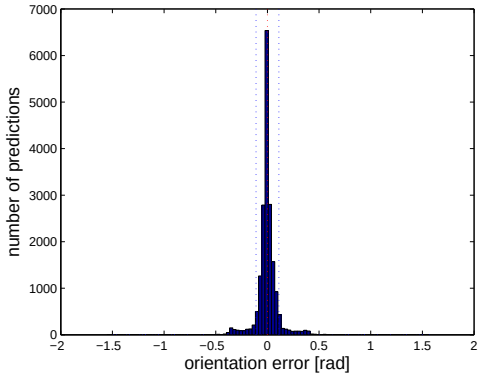
$$\mathbf{P}_m = \begin{pmatrix} 0.005 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.005 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0001 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

A.2.9 DirectedRodEMA

Distribution plots for test run in the primary model case with mean (red line) and standard deviation (blue lines):







Prediction:

$$x_p = \hat{x}_{k-1} + \hat{v}_{x,k-1} + 0.5 * \hat{a}_{x,k-1}$$

$$y_p = \hat{y}_{k-1} + \hat{v}_{y,k-1} + 0.5 * \hat{a}_{y,k-1}$$

$$\phi_p = \hat{\phi}_{k-1} + \hat{v}_{\phi,k-1} + 0.5 * \hat{a}_{\phi,k-1}$$

Update:
Before the measured angle ϕ_m (the orientation of the robot) is used, it is compared to the last orientation and turned if the difference exceeds $\frac{\pi}{2}$. As an additional measurement the *half length offset* h_m is calculated along with its EMA-smoothed estimate $\hat{h}_k$ from the current length l_m and the average diameter d.

$$h_m = \frac{l_m - d}{2}$$

$$\hat{h}_k = \alpha_h * h_m + (1 - \alpha_h) * \hat{h}_{k-1}$$

$$\hat{v}_{h,k} = \alpha_{\dot{h}} * \dot{h}_m + (1 - \alpha_h) * \hat{v}_{h,k-1}$$

$$\hat{a}_{h,k} = \alpha_{\ddot{h}} * \ddot{h}_m + (1 - \alpha_h) * \hat{a}_{h,k-1}$$

Further a more inert estimate $\hat{\zeta}_k$ of the angle is calculated:

$$\hat{\zeta}_k = \alpha_\zeta * \phi_m + (1 - \alpha_\zeta) * \hat{\zeta}_{k-1}$$

State estimates:

$$\hat{x}_k = \alpha_{pos} * x_m + (1 - \alpha_{pos}) * \hat{x}_{k-1}$$

$$\hat{y}_k = \alpha_{pos} * y_m + (1 - \alpha_{pos}) * \hat{y}_{k-1}$$

$$\hat{v}_{x,k} = \alpha_{vel} * \dot{x}_m + (1 - \alpha_{vel}) * \hat{v}_{x,k-1}$$

$$\hat{v}_{y,k} = \alpha_{vel} * \dot{y}_m + (1 - \alpha_{vel}) * \hat{v}_{y,k-1}$$

$$\hat{\dot{\phi}}_{y,k} = \alpha_{\dot{\phi}} * \dot{y}_m + (1 - \alpha_{\dot{\phi}}) * \hat{\dot{\phi}}_{y,k-1}$$

$$\hat{a}_{x,k} = \alpha_{acc} * \ddot{x}_m + (1 - \alpha_{acc}) * \hat{a}_{x,k-1}$$

$$\hat{a}_{y,k} = \alpha_{acc} * \ddot{y}_m + (1 - \alpha_{acc}) * \hat{a}_{y,k-1}$$

$$\hat{a}_{\phi,k} = \alpha_{\ddot{\phi}} * \ddot{\phi}_m + (1 - \alpha_{\ddot{\phi}}) * \hat{a}_{\phi,k-1}$$

Orientation:

$$\hat{\phi}_k = \alpha_{\phi} * \phi_m + (1 - \alpha_{\phi}) * \hat{\phi}_{k-1}$$

The orientation is turned if either $\hat{h}_k$ passes through zero or the angle estimate $\hat{\zeta}_k$ and the current velocity clearly contradict each other (the decision is based on a threshold). The algorithm stops using the $\hat{h}_k$ and $\hat{\zeta}_k$ estimates if the robot is in an upright position for more than three frames ($\frac{l_m}{d} < 1.1$). Consult the code for more details.

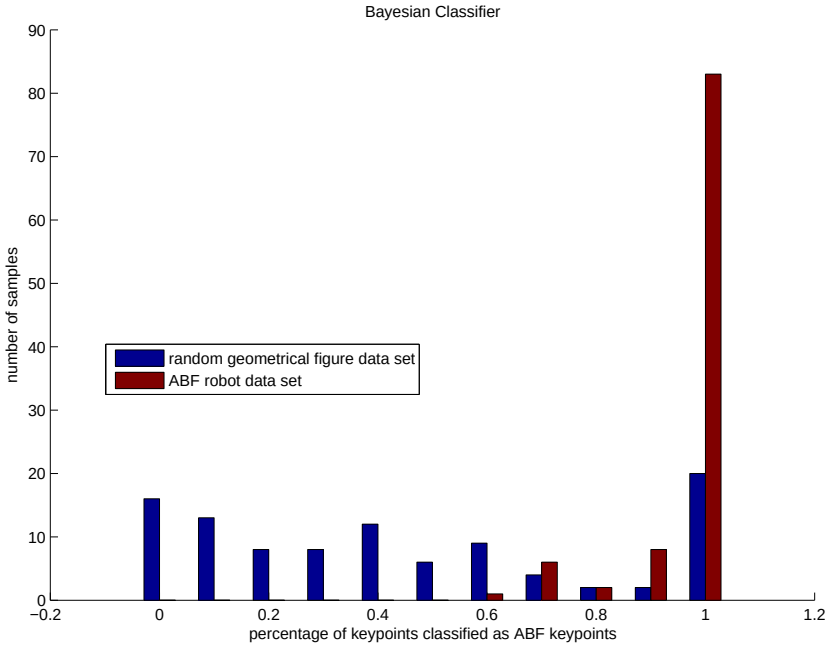
Standard parameters:
$\alpha_{pos} = 1, \alpha_{\phi} = 1, \alpha_{vel} = 0.08, \alpha_{\dot{\phi}} = 0.08, \alpha_{acc} = 0.08, \alpha_{\ddot{\phi}} = 0.08, \alpha_h = 1, \alpha_{\dot{h}} = 0.08, \alpha_{\ddot{h}} = 0.02, \alpha_{\zeta} = 0.3$

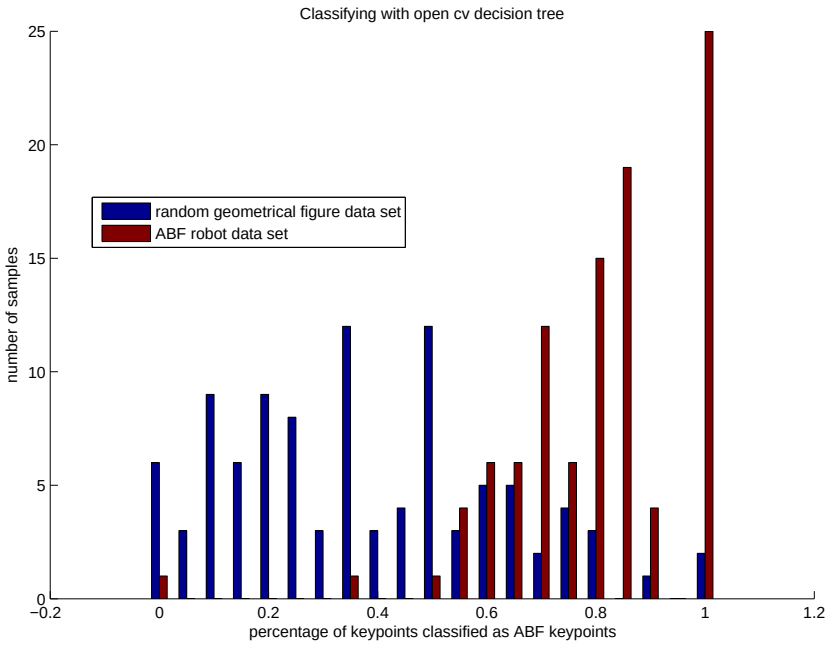
A.3 Further classification evaluation results

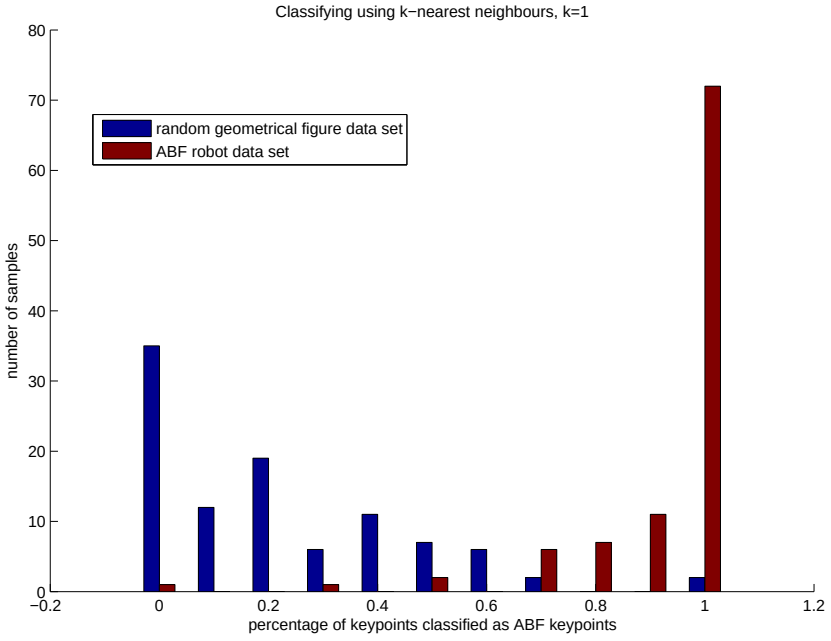
A.3.1 Falsely classified artificial feature vector percentages for classifier and extractor/descriptor pair combinations

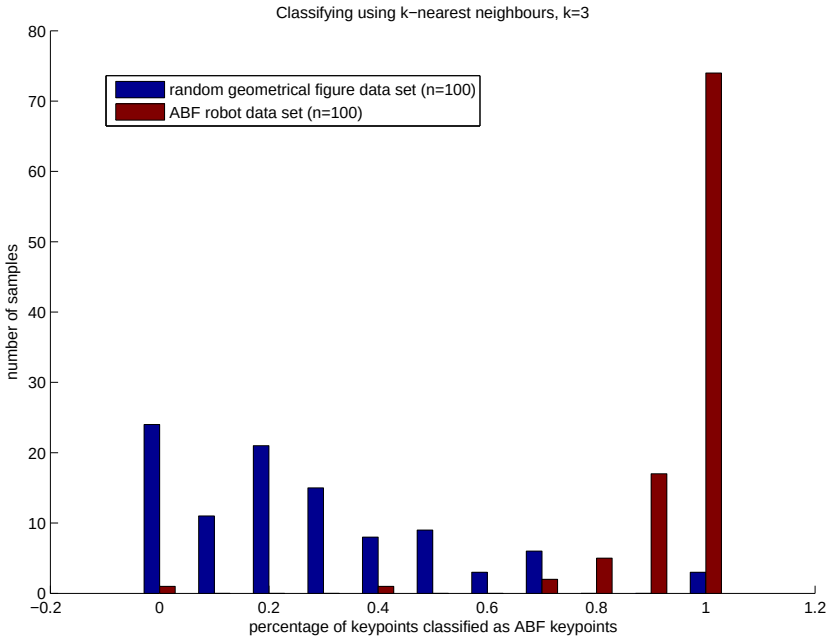
classifier type	FAST/BRISK	SURF/SURF	BRISK/BRISK	BRISK/FREAK
bayesian	25.86%	3.22%	9.30%	40.51%
k-nearest (k=1)	20.88%	7.03%	6.98%	51.90%
k-nearest (k=3)	22.42%	6.00%	5.81%	
k-nearest (k=5)	22.56%	5.56%	1.16%	
k-nearest (k=8)	29.01%	7.61%	2.33%	
k-nearest (k=10)	28.31%	7.47%	2.33%	
svm	0%	9.37%	0%	0%
decision trees	43.44%	23.57%	1.16%	36.71%
boosting	21.03%	7.91%	10.47%	34.12%
random trees	32.01%	7.17%	1.74%	26.58%

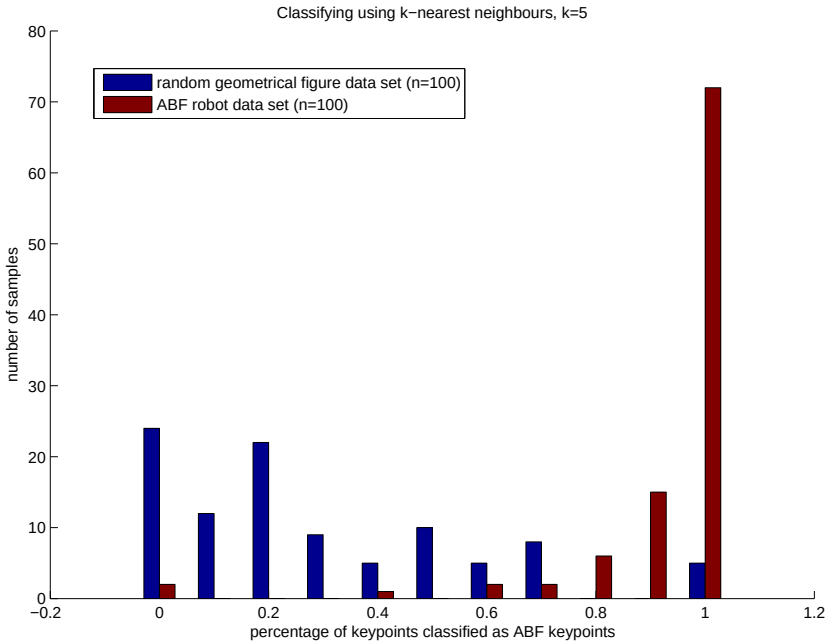
A.3.2 Type distinction capability evaluation histograms for classifiers based on SURF descriptors

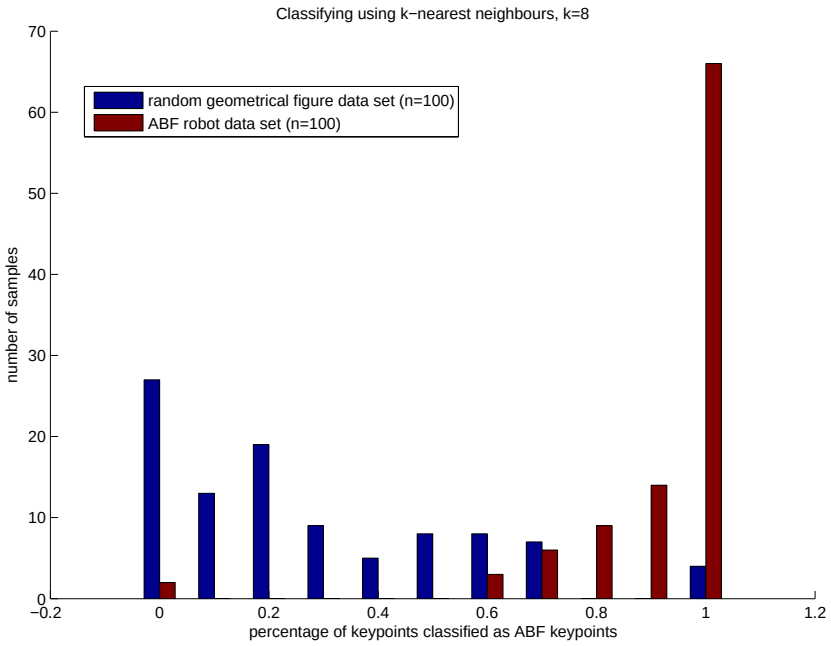


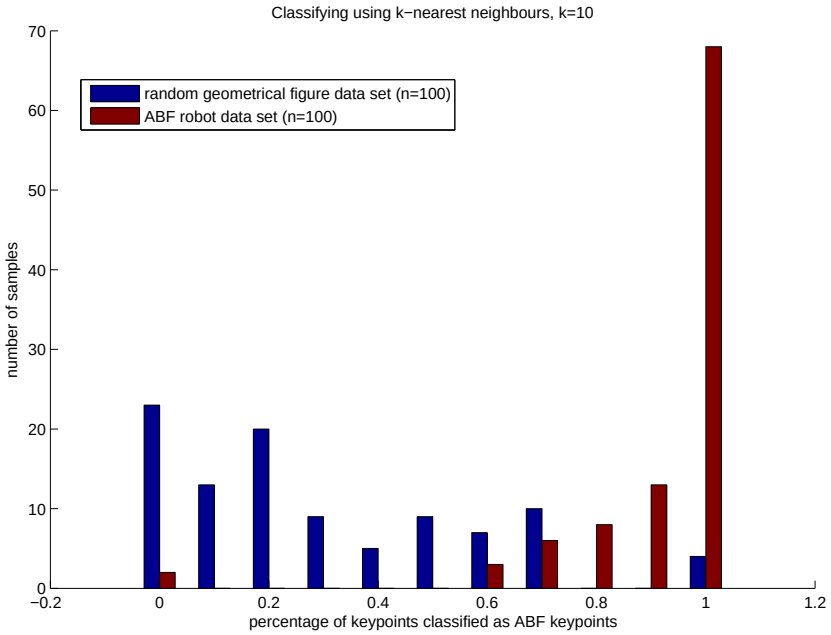

Classifying with open cv decision tree
random geometrical figure data set
ABF robot data set
number of samples
percentage of keypoints classified as ABF keypoints
0
5
10
15
20
25
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

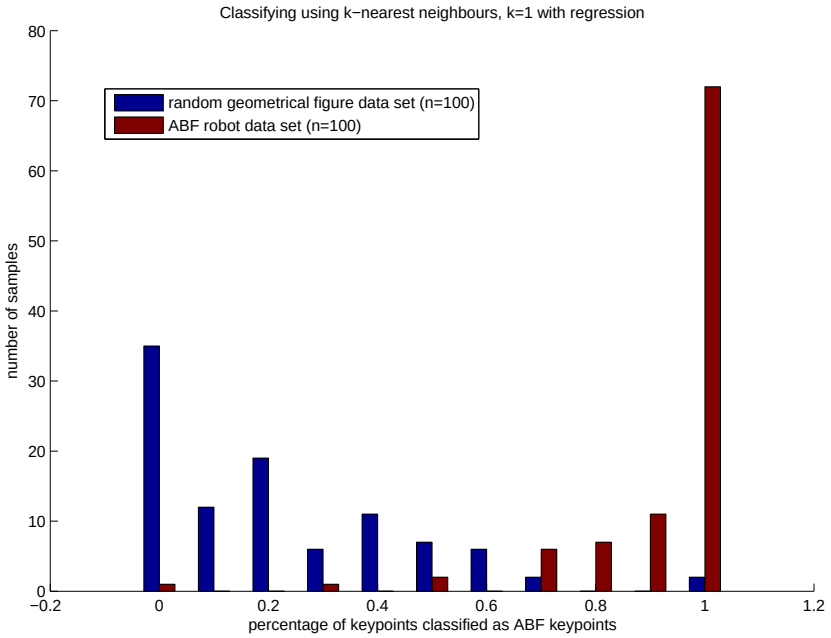

Classifying using k−nearest neighbours, k=1
random geometrical figure data set
ABF robot data set
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
80
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

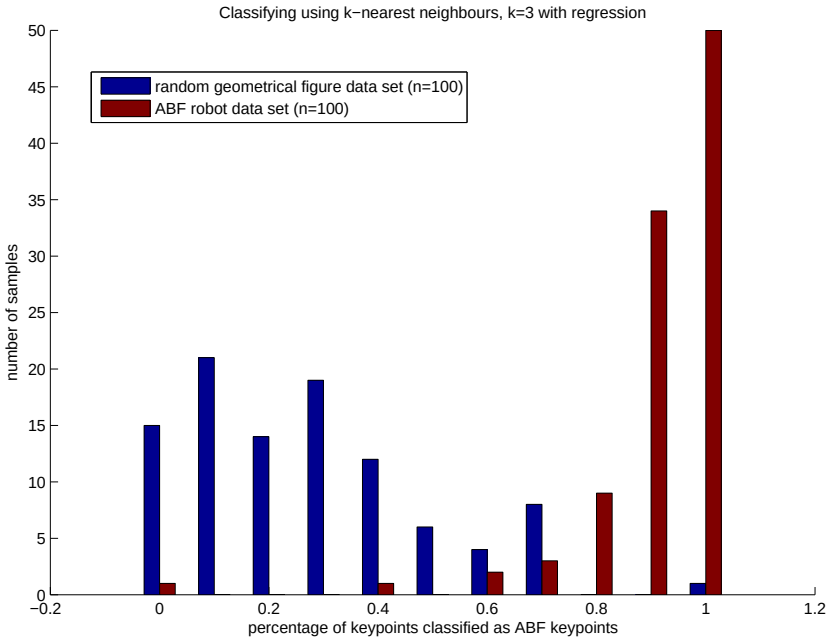

Classifying using k−nearest neighbours, k=3
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
80
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

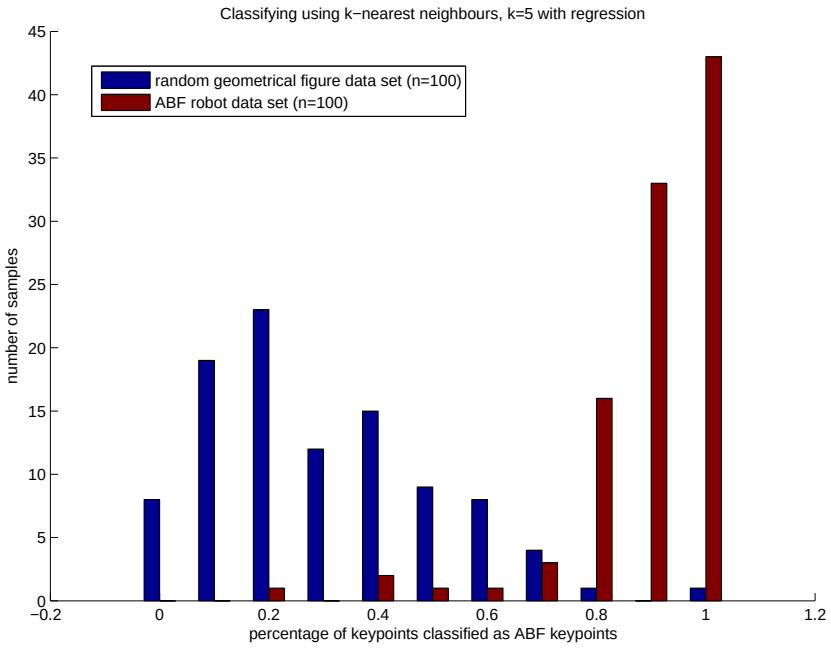

Classifying using k−nearest neighbours, k=5
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
80
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

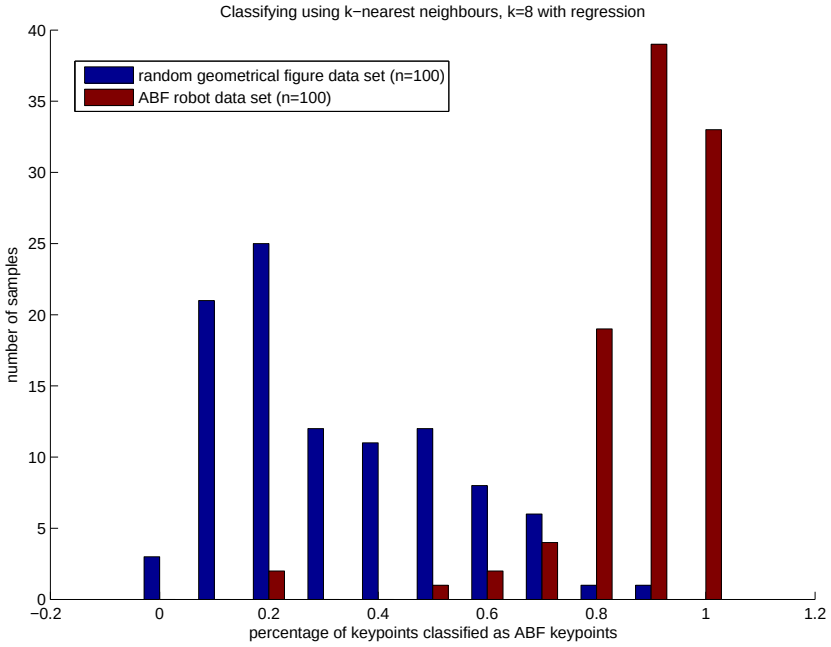

Classifying using k−nearest neighbours, k=8
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

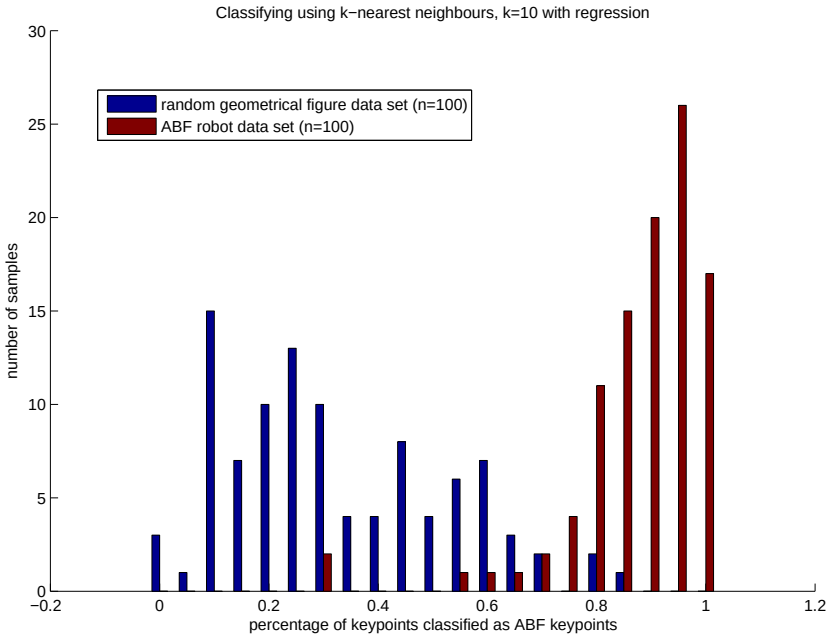

Classifying using k−nearest neighbours, k=10
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

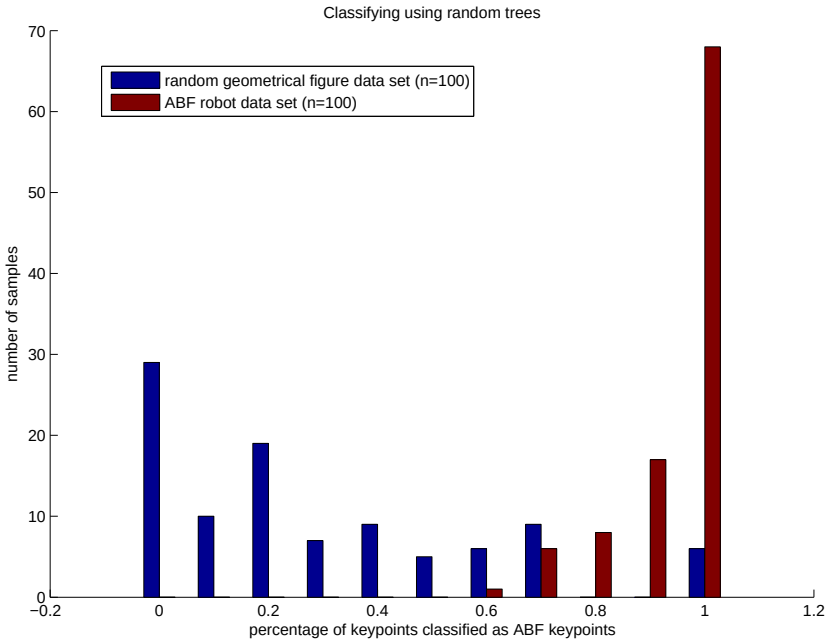

Classifying using k−nearest neighbours, k=1 with regression
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
80
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

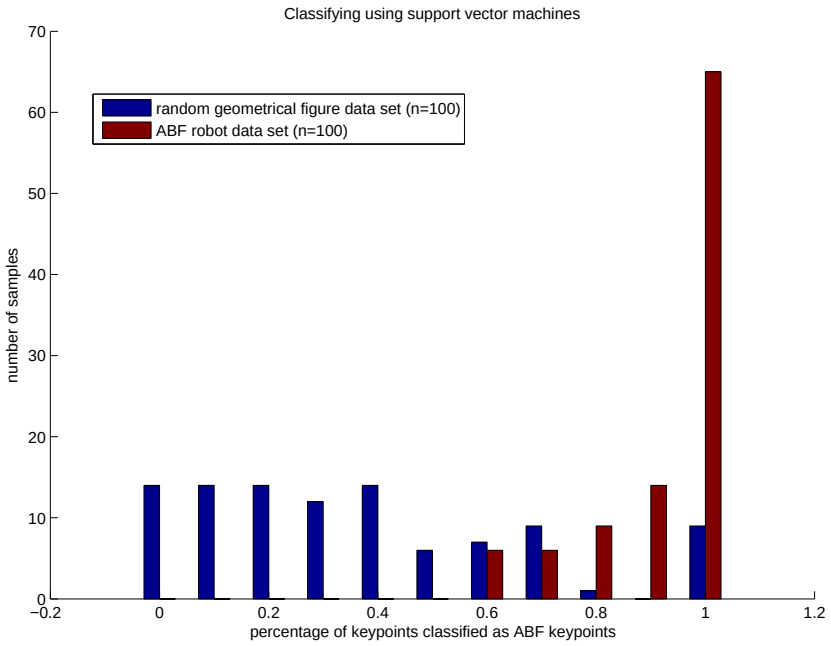

Classifying using k−nearest neighbours, k=3 with regression
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
5
10
15
20
25
30
35
40
45
50
−0.2
0
0.2
0.4
0.6
0.8
1
1.2


Classifying using k−nearest neighbours, k=5 with regression
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
5
10
15
20
25
30
35
40
45
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

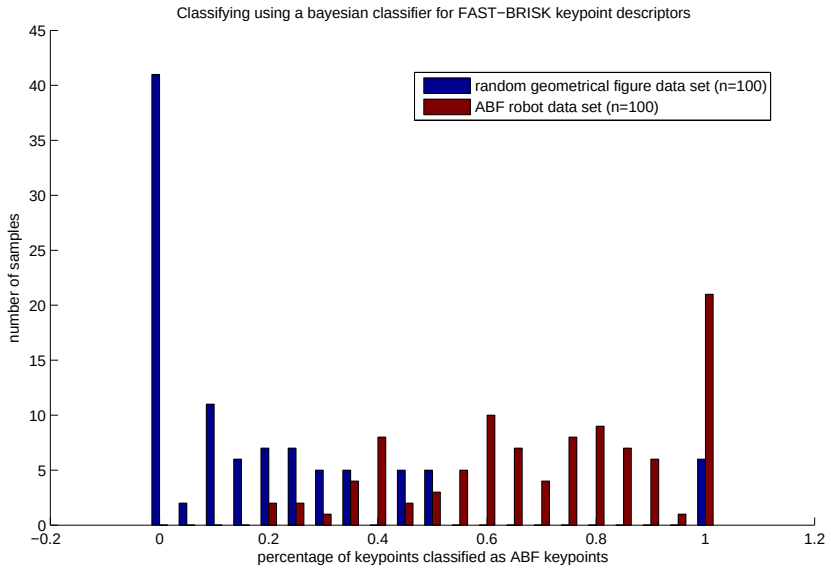

Classifying using k−nearest neighbours, k=8 with regression
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
5
10
15
20
25
30
35
40
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

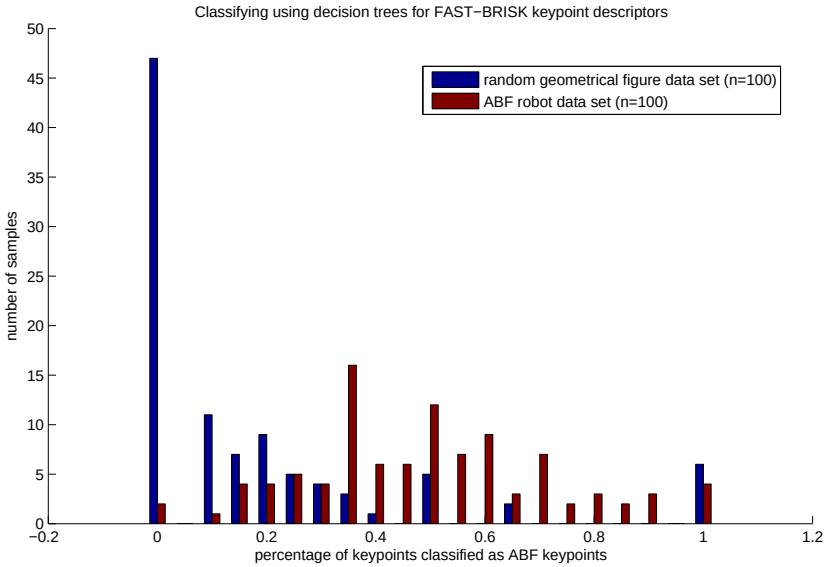

Classifying using k−nearest neighbours, k=10 with regression
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
5
10
15
20
25
30
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

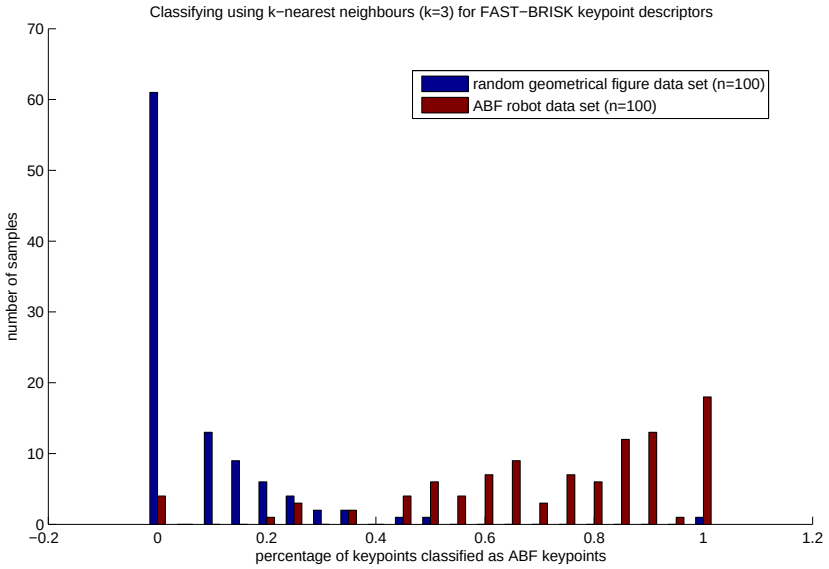

Classifying using random trees
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

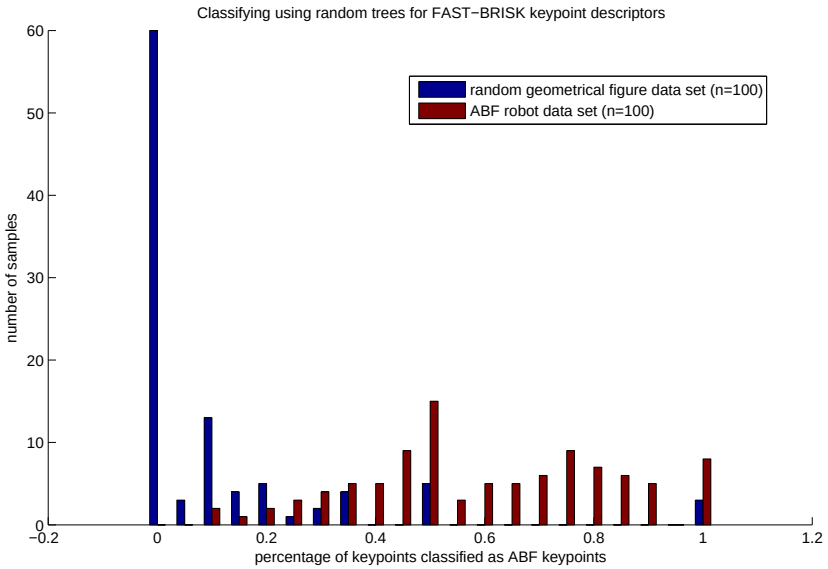


A.3.3 Type distinction capability evaluation histograms for classifiers based on FAST/BRISK descriptors



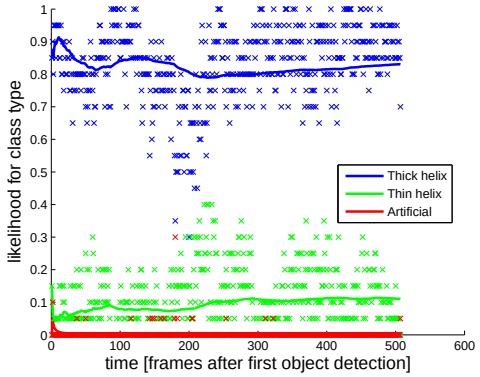

Classifying using decision trees for FAST−BRISK keypoint descriptors
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
5
10
15
20
25
30
35
40
45
50
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

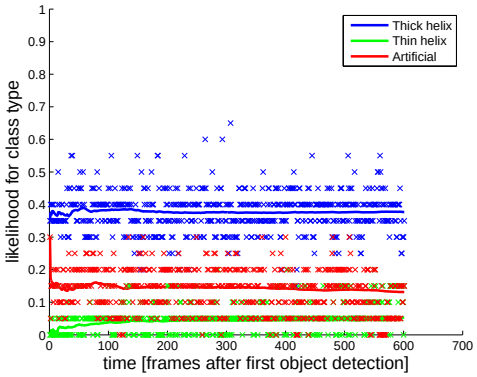

Classifying using k−nearest neighbours (k=3) for FAST−BRISK keypoint descriptors
random geometrical figure data set (n=100)
ABF robot data set (n=100)
number of samples
percentage of keypoints classified as ABF keypoints
0
10
20
30
40
50
60
70
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

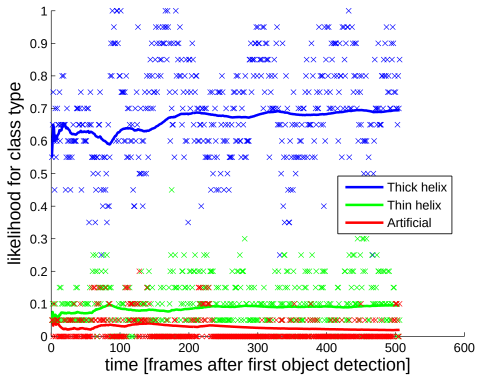


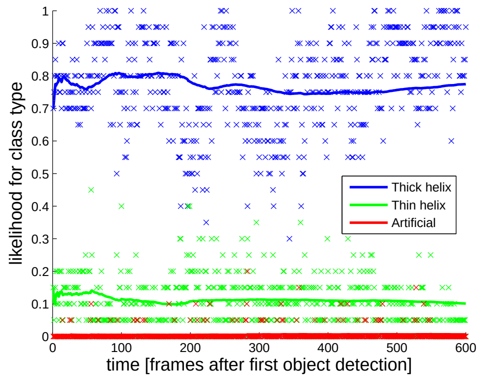
A.4 Classification results for model cases

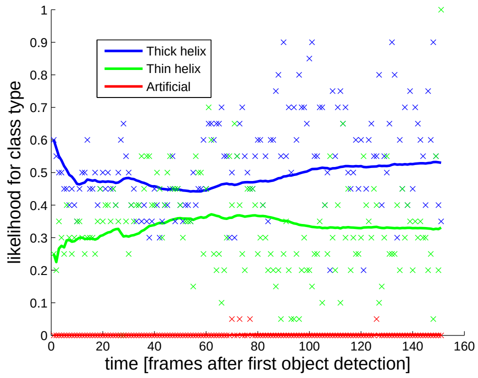
A.4.1 Secondary model case 1

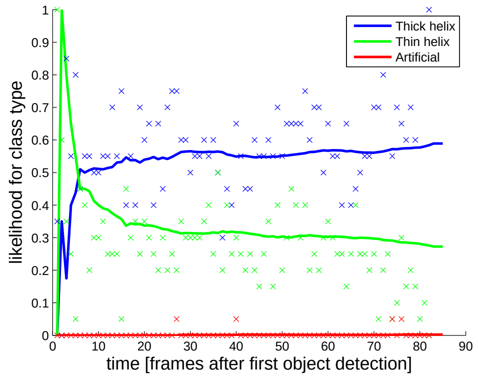


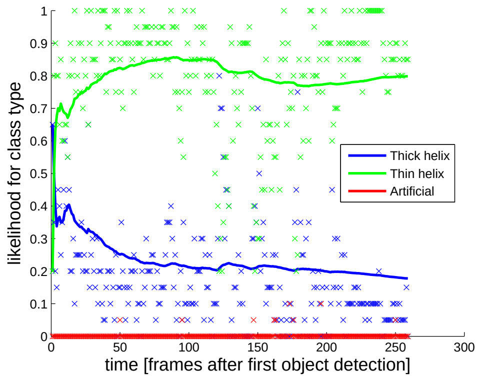


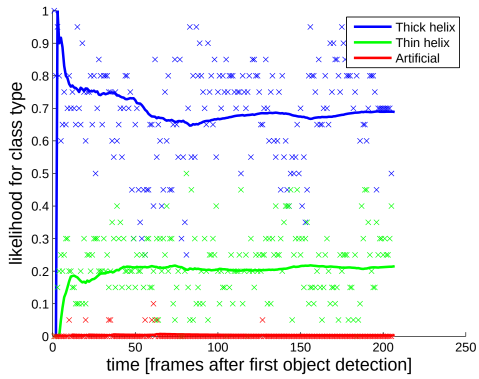

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

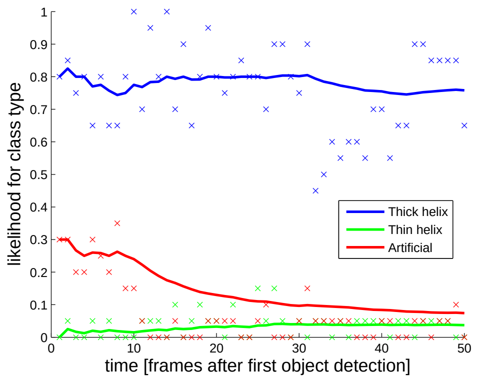

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

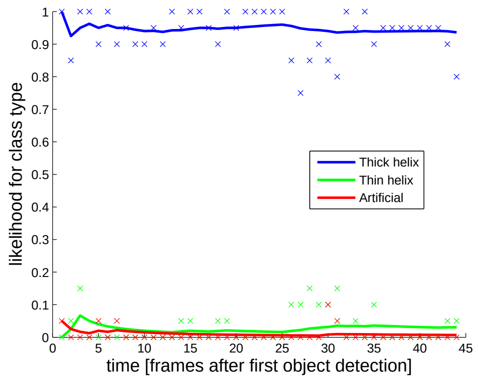

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial


likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

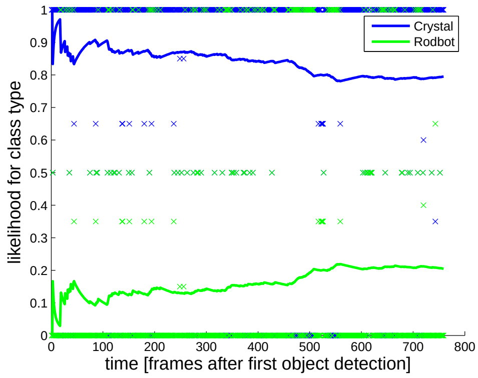

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

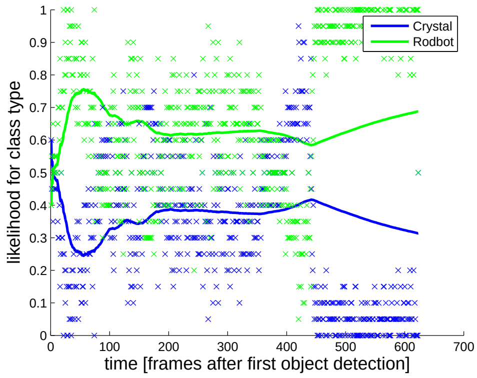

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

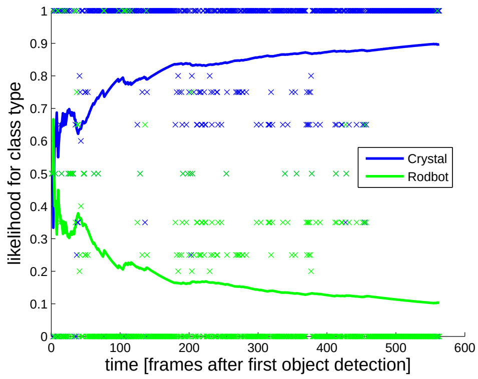

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

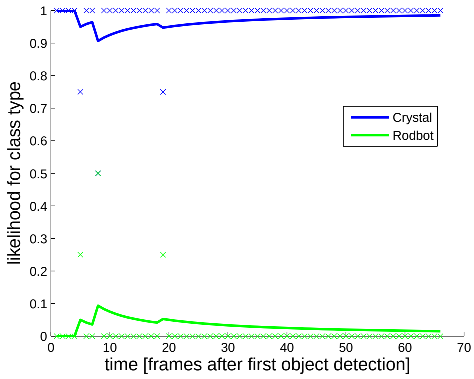

likelihood for class type
time [frames after first object detection]
Thick helix
Thin helix
Artificial

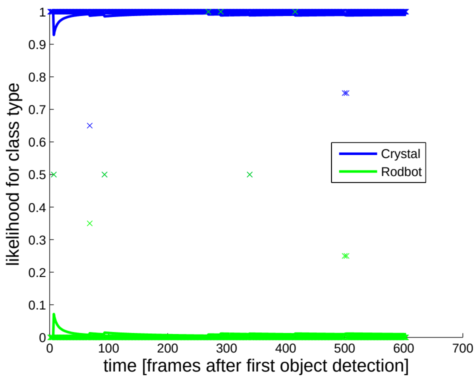
A.4.2 Secondary model case 2 with additional features











A.4.3 Secondary model case 2 without additional features