Czech Technical University in Prague &
Luleå University of Technology

Faculty of Electrical Engineering
Department of Cybernetics

MASTER'S THESIS

A semantic interpreter for multimodal and multirobot data

Philipp Florian Käshammer

supervised by
doc. Ing. Tomáš Svoboda, PhD.

August 15, 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Philipp Käshammer**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Diploma Thesis: **A semantic interpreter for multimodal and multirobot data**

Guidelines:

Multiple ground and aerial robots capture various data during their missions. The data are tagged and stored in a low-level database. The goal of the thesis is to use the very recent developments [1],[2] and implement a ROS [3] node that connects the deep-nets to the TRADR [4] system. The node crawls through the data in the low-level database and anything recognized pushes to a high-level database properly annotated.

1. Install a TRADR compatible system, download relevant bag-data for experiments
2. Learn the interfaces of the low and high level TRADR databases
3. Implement ROS nodes for crawling the low-level database feeding the data in the object detection engine
4. Write the annotated data into the high level database

Bibliography/Sources:

[1] Girshick, Ross and Donahue, Jeff and Darrell, Trevor and Malik, Jitendra. Region-based Convolutional Networks for Accurate Object Detection and Segmentation. In IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015

[2] Ren, Shaoqing and He, Kaiming and Girshick, Ross and Sun, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Neural Information Processing Systems (NIPS), 2015

[3] Robot Operating System, http://www.ros.org

[4] Long Term Human Robot Teaming for Robot Assisted Disaster Response, http://www.tradr-project.eu. Various technical reports/deliverables.

Diploma Thesis Supervisor: doc. Tomáš Svoboda Ing., Ph.D.

Valid until the summer semester 2016/2017

prof. Ing. Michael Šebek, DrSc.
Head of Department

L.S.

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 24, 2016

Abstract

Huge natural disaster events can be so devastating that they often overwhelm human rescuers and yet, they seem to occur more often. The TRADR (Long-Term Human-Robot Teaming for Robot Assisted Disaster Response) research project aims at developing methodology for heterogeneous teams composed of human rescuers as well as ground and aerial robots. While the robots swarm the disaster sites, equipped with advanced sensors, they collect a huge amount row-data that cannot be processed efficiently by humans. Therefore, in the frame of the here presented work, a semantic interpreter has been developed that crawls through the raw data, using state of the art object detection algorithms to identify victim targets and extracts all kinds of information that is relevant for rescuers to plan their missions. Subsequently, this information is restructured by a reasoning process and then stored into a high-level database that can be queried accordingly and ensures data constancy.

Acknowledgements

First of all, I want to thank my supervisor Tomáš Svoboda for his great guidance, comments and for providing me with all necessary resources. Furthermore, I would like to thank Daniel Robin Reuter who introduced me to the initial state of the project and the semantic interpreter. I would also like to thank Tomáš Petříček, Martin Pecka, and Vladimír Kubelka for the assistance with the robot and answering framework related questions. Many thanks to Otakar Jašek for proofreading as well. Finally I'd like to thank my parents for their endless support during my studies, as well as the rest of my family and friends for their backup.

In addition to all those, I would like to thank the European Commission for their financial support to carry out my studies as part of the Erasmus Mundus Joint European Master in Space Science and Technology.

Declaration of Authorship

I, Philipp F. Käshammer, declare that this thesis titled, "A semantic interpreter for multimodal and multirobot data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Place and Date	Signature

Contents

List of Figures

List of Tables

List of Abbreviations

UUID Universally unique identifier

CNN Convolutional Neural Network

RMSE Root Mean Square Error

TRADR Long-Term Human-Robot Teaming for Robot Assisted Disaster Response

HLDB High-Level Database

LLDB Low-Level Database

ROS Robot Operation System

RPN Region proposal networks

ROI Region of Interest

NMS None Maximum Suppression

ROC Receiver Operating Characteristic

RDF Resource Description Framework

1 Introduction

Earthquakes, hurricanes, tsunamis or simply fire, just to name a few, are common disaster scenarios especially in the era of climate change, that can be so devastating in magnitude that they often overwhelm human rescuers. Robots however don't fatigue from multi day operations or suffer emotional distress. Therefore, equipped with advanced sensors, they are perfect assistants and have the potential to make a crucial difference when it comes to saving lives. Project TRADR (Long-Term Human-Robot Teaming for Robot Assisted Disaster Response (`http://www.tradr-project.eu`)) aims at developing methodology for heterogeneous teams composed of human rescuers as well as ground and aerial robots. [1] As the robots swarm the disaster sites they scan their environment and collect all sorts of raw data. After such a sortie the data of all robots is stored into a main low-level database.
The ultimate goal of this project is to develop and implement a semantic interpreter that will search the for humans unmanageable amount of raw data (mainly images) for potential targets and then lift extracted information about its findings into a high-level database where it can be easily accessed by rescuers to plan further operations. To accomplish this task, state of the art objected detection and recognition software, in particular `faster-rcnn` was used and integrated into the semantic interpreter. Therefore, prior to this work, its performance was experimentally evaluated on artificial disaster datasets. Moreover, in this work, a simulation pipeline was implemented for the purpose of developing and testing the semantic interpreter.
In the following the content and structure of this work is briefly presented:

- **Chapter 2** gives a brief review of `fast-rcnn's` (`faster-rcnn's` predecessor) performance results on artificial victim datasets that have been evaluated prior to this work. Furthermore, basic knowledge about state of the art convolution neural networks is provided before `faster-rcnn`'s main features of are outlined.
- **Chapter 3** aims to equip the reader with all knowledge about the TRADR framework required for the understanding of the semantic interpreter's working principles and the content provided in this work.
- **Chapter 4** is the main chapter of this work and explains the semantic interpreter in great detail. First fundamental knowledge about databases in general is given with regard to the presented work. Subsequently all methods of low-level data extraction process including object detection and recognition are illustrated. In addition to that, all aspects and methods of the high-level reasoning process are demonstrated. Finally the implementation of the simulation pipeline is discussed.
- **Chapter 5** evaluates and discusses the performance of the semantic interpreter as it was developed during this work. Therefore, the two main simulation scenarios are illustrated and the corresponding results presented. Besides that, weaknesses are outlined and improvement suggestions are made.
- **Chapter 6** simply provides a user manual of how to install and run the semantic interpreter.
- **Chapter 7** finally summarizes the whole work and derives a conclusion based on the evaluation made in the previous chapter and with respect to the original goal.

2 Previous work

The heart of the semantic interpreter is `faster-rcnn` which is a special type of a convoluted neural network (CNN)[2](https://github.com/rbgirshick/py-faster-rcnn). It was mainly chosen for its extremely fast performance. There are tow reasons for that: First, multiple robots crawling and recording a disaster side with multiple cameras can produce a huge amount of data in little time. As the semantic interpreter has to process at least most of those images and with object detection being a computational expensive task, the detection speed is of importance. Although the semantic interpreter is not supposed to run in real time on each robot but rather offline on a central main computation unit, this is still true. Second, TRADR's ultimate goal is to rescue victims of disastrous events. Therefore, successfully detecting and localizing targets is essential to accomplish this task. Whether a victim is detected or not could therefore make the difference between life and death.
For those reasons and also because disaster-site data naturally differs from conventional training data, the performance of the `fast-rcnn` object detection algorithm was analysed on an artificial victim dataset in preparation to this work. Therefore, the most important results are restated in section 2.2 below. The semantic interpreter however uses `fast-rcnn`'s improved and newer version called `faster-rcnn`. In the following its features are briefly explained.

2.1 Detection using faster-rcnn

In particular, the `Caffe` network model VGG_CNN_M_1024 was used, which is capable of detecting and classifying objects of 20 different classes. It was pre-trained using `faster-rcnn` and provided by their authors as well.[2]
Since this work is mostly concerned with the implementation of the semantic interpreter with the corresponding data extraction, reasoning process and simulation pipeline, the detection algorithm will be treated rather as a black box that is plugged into the interpreter. However a few basic features and concepts that are of relevance for this work are explained in the following:

2.1.1 Convolutional neural networks (CNNs)

Convolutional neural networks are a special type of so called artificial neural networks (ANNs). They are statistical learning algorithms that are inspired by natural neural networks like the human brain. Similar to that, they can be viewed as a system of interconnected neurons that propagate information given as the input through the network, in order to compute a output. The amount of inputs is usually large compared to the output. This is because each neuron computes a single output for multiple inputs and the same amount of weights (inner product). The weights can be adjusted by different machine learning algorithms (e.g. backward-propagation in supervised learning) until the network produces the desired output for a given input. [3] [4]
Neural networks are usually composed of different layers that serve different purposes. CNNs however are special in that they are composed out of specific types of layers. The most important are therefore briefly explained in the following.

2.1.2 Layers

Convolutional neural networks are mainly composed of three different layer-types. Those are:

- **Convolutional layers** are the core building blocks of a CNNs. Such a layer receives a single 2-dimensional input, usually the feature map computed by the previous layer or a image. It then computes multiple feature maps as an output by convolving filters across the input feature map (one output feature map for each filter). Figure 2.2 provides a visualization. The filters can be seen as the parameters of a convolutional layer and are adjusted during training. [5] [6] [4] [3]
- **Max-pooling layers** are mainly used for dimensional reduction of propagating feature maps and often present after a convolutional layer. There are many different pooling layer types. Here only the most commonly used max-pooling layer is explained. Neurons in this layer take rectangular sub-regions of 2-dimensional input feature maps and find the maximum among all elements within that sub-region which in turn is send to the output. This is operation is illustrated by figure 2.1. [3][6]
- **Fully connected layers** perform high-level reasoning. As the name suggests a fully connected layer connects all outputs of the previous layer to all neurons present in the current layer (figure 2.2). Since their output does not form a rectangular grid, they are often followed by layers of the same type instead of convolutional and max-pooling layers. That is also why they normally occur at the end of a network. [3][4]

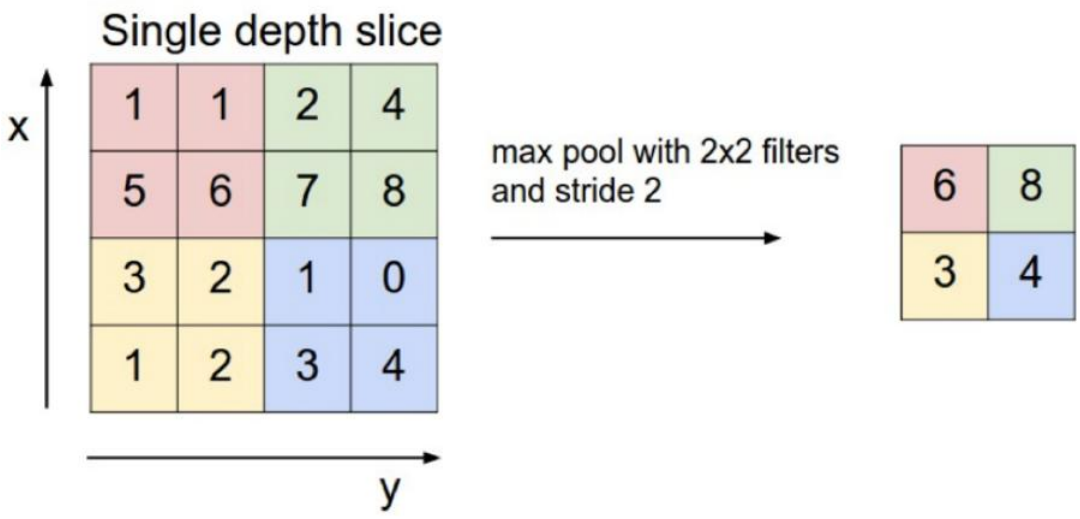


Figure 2.1 Illustration of max pooling. Provided by [7]

2.1.3 Feature vectors

Feature vectors exist for every layer of a CNN. They are simply the output of a specific layer that can also be located in the middle of the network. It is therefore dependent on all previous layers in the network but not on any of the following ones. Their length depends on the structure of the neural network and on the layer itself for which the features are extracted.
In this work, feature vectors are extracted for the last two layers of the `VGG_CNN_M_1024` network model and then used to compare similarity between multiple detection. Section 4.2.1 provides more detailed information about that.

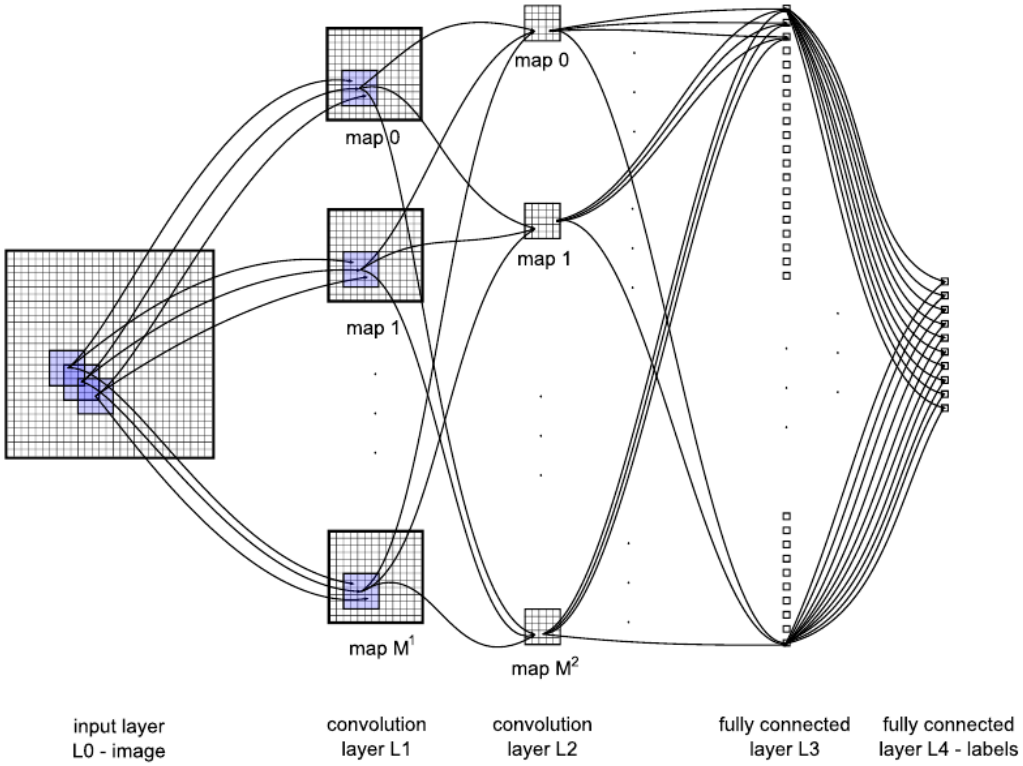


Figure 2.2 Architecture of a convolutional neural network. In this case, the convolutional layers are fully connected. Both convolutional layers use a kernel of 5 x 5. Provided by [4]

2.1.4 Region proposal network (RPN)

In order to recognize and classify objects in images, they have to be identified as a region of interest (ROI) within the scene of the image first. This task is referred to as object detection. Here lies the main difference and improvement of `faster-rcnn` compared to it's predecessor. `Fast-rcnn` uses the region proposal algorithm "selective search" that is executed prior to the classification task. Those regions of the image are then cropped and fed to the network as an input. Figure 2.3 gives a illustration of that. In contrast to that, `faster-rcnn` uses so called region proposal networks. Here the whole image is fed to the network at once and only then, after convolution and pooling was applied, region of interests are generated based on the size reduced feature map. Those ROIs are then fed to the fully connected layers. Mainly due to the reduced size of the feature map compared to the full image, `faster-rcnn` is able to compute region proposals a lot faster compared to the classical approach.
Although the region proposal method increases detection speed a lot, the classification performance does not suffer. Consequently it can be assumed that the evaluation results for `fast-rcnn` described in the following section also apply to `faster-rcnn` at worst. [2] [6]

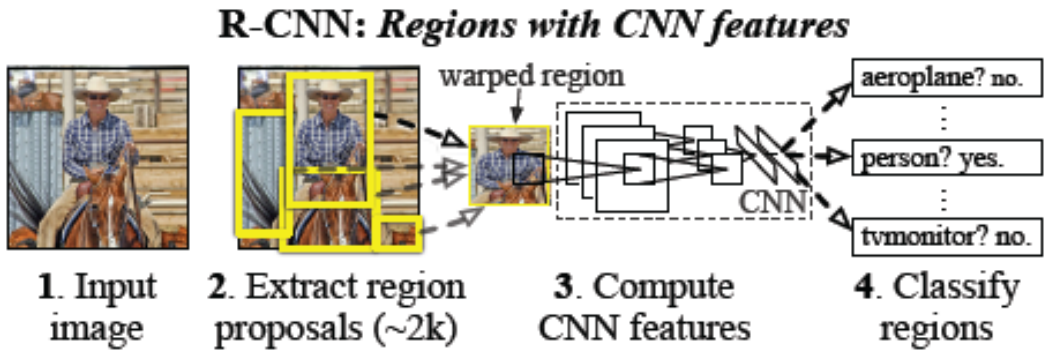
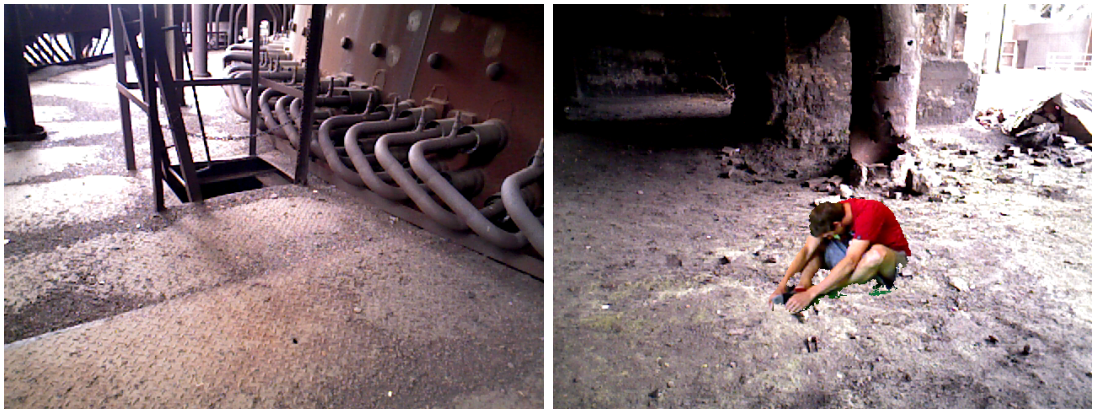


Figure 2.3 Object detection system overview. The system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional network (CNN), and then (4) classifies each region using class-specific linear SVMs. Provided by [2]

2.2 Performance analysis on artificial victim dataset

In the following, the main evaluation results are shortly restated without a detailed explanation. However, the full information about conducted experiments and results can be found here: (`https://gitlab.fel.cvut.cz/students/kashammer-phillip/blob/master/idp/report/report.pdf`)[8]
Performance analysis of `fast-rcnn` was done by feeding the algorithm with artificial disaster datasets and evaluating the receiver operation characteristic (ROC) graph which is generated by plotting the true-positive-rate (TPR) against the false-positive-rate as the discrimination threshold varies. Examples of artificial victim images are illustrated by figure 2.4. Three similar experiments have been conducted which only differ in the composition of the input dataset. For the purpose of this work, it is enough to refer the reader only to experiment number three as there the most realistic input dataset was used which contains the same amount of background images (negatives) as victim images (positives).
The resulting ROC curve is depicted green in image 2.5. It implicates a reasonable good performance with an area under the curve (AUC) of 0.8973, while a value of 1.0 would describe a perfect binary classifier. In contrast, a value of 0.5 would describes a classifier with a 50% chance of making the right decision. Thus, it would be useless. The result lead to the conclusion that the satisfactory performance of `faster-rcnn` also holds for victim detection in unconventional disaster surroundings. Also it should be mentioned here that the detection result can be improved by partly retraining the neural network with real victim data, once a sufficient amount exists.[6]



a) Background Example **b)** Victim Example

Figure 2.4 Example images of the artificial disaster dataset

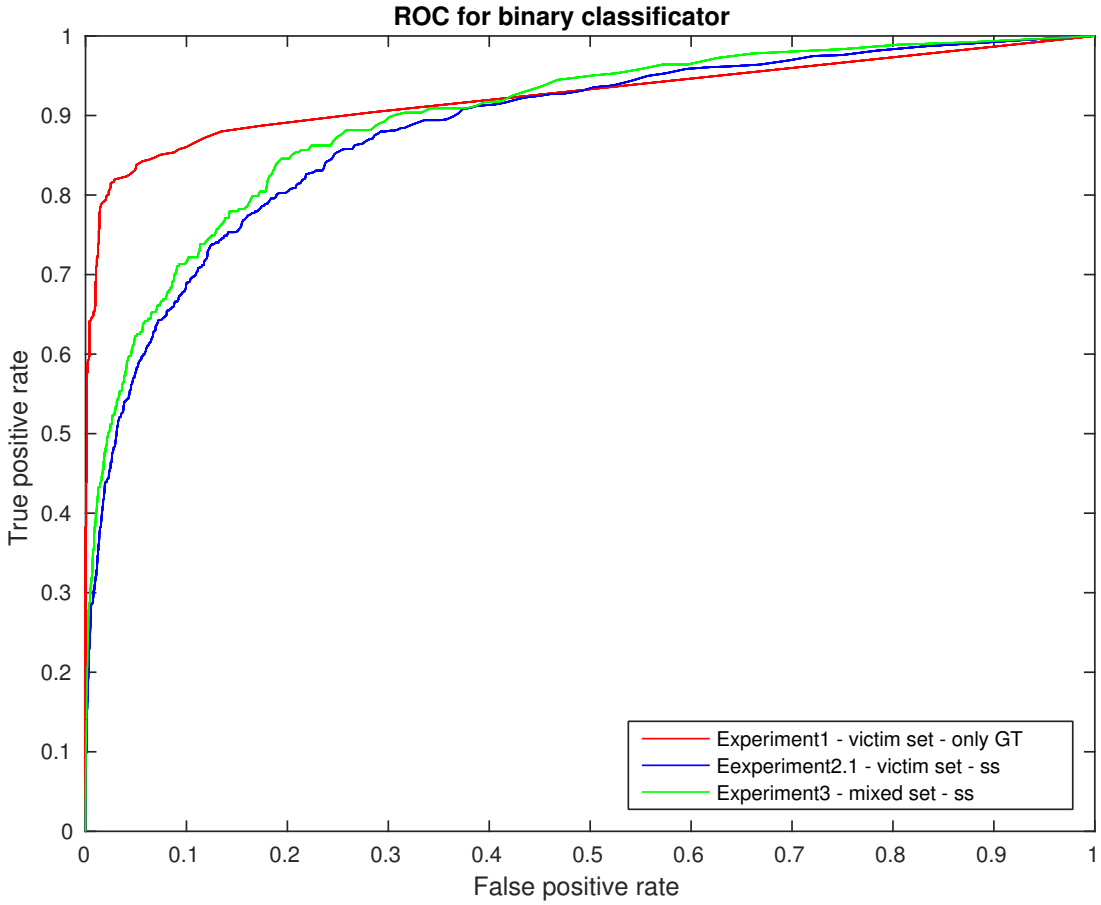


Figure 2.5 Resulting ROC curves of conducted experiments as labelled in the legend. Graphs are obtained using Matlab.

3 TRADR framework

"The TRADR system enables humans and robots to work as a team, exchange information and operate together to accomplish complex rescue scenarios." It is therefore composed of many hardware (e.g. ground and aerial robots) and software elements.
"The robot operationg system (ROS) is used as a middleware-framework in TRADR to facilitate reuseable, decoupled modules for an aggregated and integrated system."[1]
This section aims to provide preliminary information about those components that are relevant for understanding of the here presented work. Those are in particular the TRADR ground rover (section 3.1) and ROS (section 3.2).

3.1 The ground rover

In order to explain the functionality of the semantic interpreter, who's task is to extract highly relevant information from the raw data that was acquired by a swarm of robots, knowledge about the robot's sensors and data acquisition is required.
The semantic interpreter in this work was developed only using data recorded by the TRADR ground rover shown in figure 3.1. As depicted, the robot is equipped with a `ladybug3` - 360° camera and a `SICK LMS-151` laser-scanner mounted at the front and with a field of view of 270°.[9][10] For the detection of victims, which is one of the core task of the semantic interpreter not all six camera frames are used but only camera frame 1, 2, 3 and 5 as labelled in the figure 3.1. Camera frame 4 is pointing backward and its field of view is blocked by other sensors. Camera frame 6 is pointing upwards and therefore useless for victim detection. Moreover, the lasers field of view is overlapping with the camera's field of view only for the four cameras frames listed before. This is relevant as the point-cloud acquired by the robot provides depth information that is used for target localization later as described in section 4.2.2. However this is not completely true for camera frame 3 and 5. Consequences are discussed in section 5.2.5.

3.2 The Robot Operating System (ROS)

As stated above, TRADR uses the Robot Operating System (ROS) which is a flexible framework for writing robot software. It provides a collection of tools, libraries, and conventions with the goal to simplify the task of creating complex and robust robot behaviour. ROS not only organizes the control and operation of the robots but also the databases used by the semantic interpreter (see section 4.1). ROS is using three main logical components: nodes, topics and messages.[11] In the following, those and a view more core features of the powerful architecture are briefly explained:

- **ROS-Nodes** are simply executables that use the ROS client library. Nodes communicate with each other by exchanging ROS-messages over the publisher/subscriber middleware-architecture based on ROS-topics. [11]

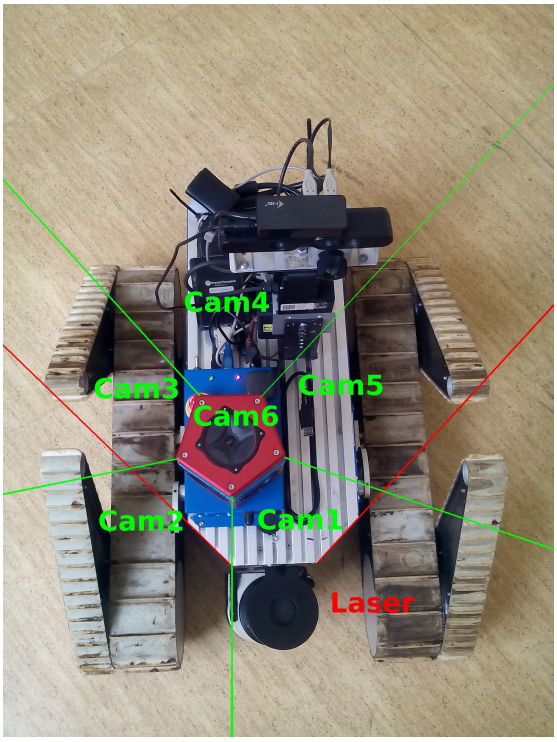


Figure 3.1 Picture of the TRADR ground rover with illustration of the cameras' (green) and laser's (red) field of view. Note that the angles are not accurate and only serve as illustration.

- **ROS-Messages** are a simple data structures that are comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported as well as arrays of them, even in arbitrarily nested structures. ROS also provides a set of predefined messages, however, new ones can easily be created to fit individual purposes. [11]
- **ROS-Topics** are named buses over which nodes exchange messages. Multiple nodes can therefore register as publishers or subscribers of the same topic. Every time a node publishes a message to a topic, all the nodes that subscribe to that topic receive the message. Therefore nodes are not aware of who they are communicating with.[11]
- **ROS-Bag** is file format in ROS named after its .bag extension. ROS-bags are basically recordings of the whole system, storing all ROS-messages that were published during recording time. Afterwards, ROS-bags can be played again and thus simulating the whole system.[11]
- **ROS-Transformations** are simply coordinate transformations between two different coordinate frames. ROS-transformations can be buffered and looked up again later for specific frames and times.[11]

4 The semantic interpreter

A semantic interpreter links low-level raw data (e.g. images) to high-level percepts (e.g. victims, detections or locations). Low- and high-level data is therefore organized by and stored in specific software tools called "databases". In other words, the interpreter connects a low-level database to a high-level database. Both provide different functionality and serve different purposes. Those are explained in detail in the following section 4.1. The word "semantic" is defined as "relating to the meanings of words and phrases"[12]. In that sense, a semantic interpreter's main purpose is to crawl through the for humans unmanageable amount of raw image data, while extracting and refining meaningful information first. This information is then stored into the high-level database, in a format, that is easy for human rescuers to query and visualize, so that they can plan rescue missions accordingly. The predefined ontology, a directed graph structure, is necessary to determine which high-level concepts can exist and also, in what kind of relation these can stand to each other. Note also that multiple semantic interpreters can exist in parallel. Each is then responsible for handling a certain type of low-level raw data and corresponding high-level objects.
In this main chapter the semantic interpreter's structure, algorithms as well as its simulation environment are explained in detail. After introducing properties, functionality and purpose of both databases, consecutively, the low-level side of the interpreter is explained including victim detection, feature vector extraction and victim localization (section 4.2). On top of that, the high-level side of the interpreter is illuminated, describing how new detections are processed and assigned to a specific victim (section 4.3). The final section 4.4 of this chapter is providing a closer look on the simulation pipeline which differs from the real case scenario in that it imposes timing constraints on the interpreter. This is inevitable, as project TRADR is still in a developmental stage and required features have to be implemented first.

4.1 Databases

In order to demonstrate how the interpreter is managing data and to explain the main reasoning process, a fundamental understanding of a database is required and therefore provided to the reader first. Furthermore, differences between the low-level and high-level database are outlined in this section.
Project TRADR is still in its developing stage. Therefore, as a design choice, both databases are deployed in two separate virtual machines, so called "Docker Container". This provides a certain hardware architecture independence and also enables developers to replace a database without much complication if necessary.[13]

Definition: Database. "Often abbreviated DB, a database is basically a collection of information organized in such a way that a computer program can quickly select desired pieces of data. You can think of a database as an electronic filing system."[14]

Definition: Database management system. "A database management system (DBMS) is a computer software application that interacts with the user, other applica-

tions, and the database itself to capture and analyse data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases." [15]

4.1.1 The low-level database (LLDB)

In general, the purpose of the LLDB is to store all kinds of raw sensor data like images, point-clouds and depth information that describe the robots surroundings, but also housekeeping data like accelerations, position and coordinate-frame transformations. In other words: Information about the robot itself. That is why each robot has its own low-level database on board. Once the robots return from a sortie to their stations, the acquired data is downloaded and merged into a main LLDB. For that reason it is necessary to attach meta-information like time-stamp and origin to every data package that is saved. Also, the main database is not located on any robot but on a stationary computer that provides more powerful computation resources, e.g. a strong graphics processing unit (GPU) to accelerate detection.
To the extent of this project, the open-source database *MongoDB* is used for low-level purposes. Its database management system is classified as *NoSQL*.[16]
"*MongoDB* avoids the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas, making the integration of data in certain types of applications easier and faster."[16]
As just stated, its main features are among others speed and easy usage as no sophis-

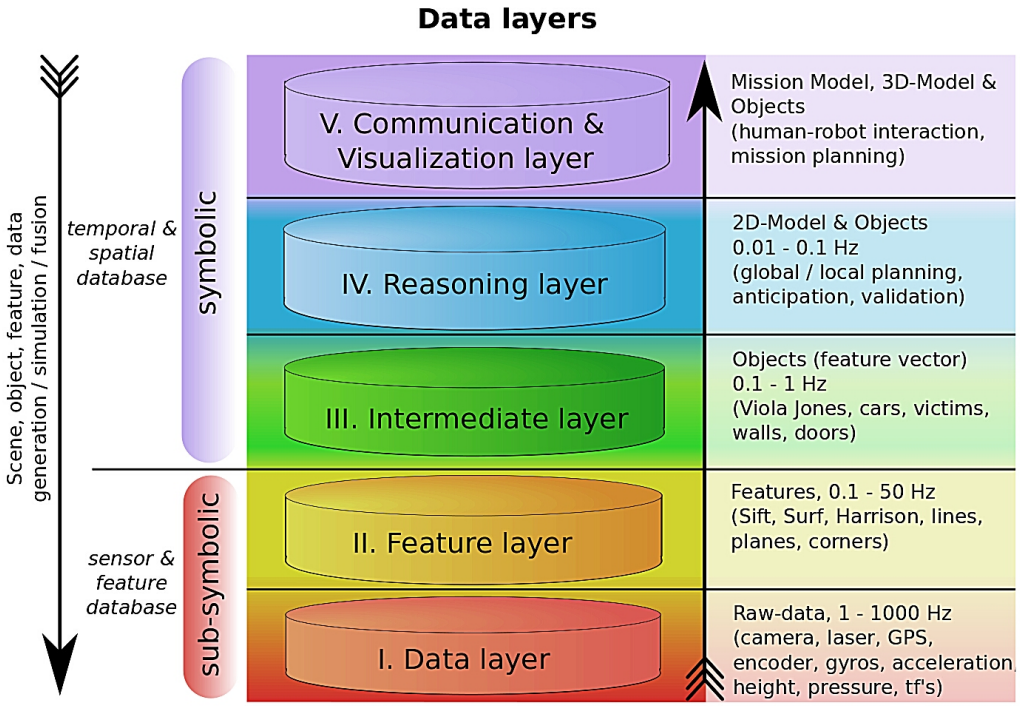


Figure 4.1 Scheme depicting the data-layers of TRADR's databases.[13] Note that this scheme was coined at the beginning of the TRADR project. Thus, the actual structure might differ from that by now.

ticated query language is needed but rather a simple key in the form of a universally unique identifier (UUID), by which the according data can be extracted. The LLDB can therefore considered to be a simple dictionary. Speed is of relevance because some sensors can provide data at high rates up to 1000Hz. Figure 4.1 depicts a scheme of TRADR's data layers. The actual structure might differ from that, however, it serves well to illustrate the differences between both databases with respect to sensor rates and content. The lower two layers marked as *sub-symbolic* can be regarded to represent the low-level database, whereas the upper three, marked as *symbolic*, represent the high-level database.

4.1.2 The high-level database (HLDB)

The high-level database is more complex than its low-level opponent as it is used for all kinds of abstract reasoning. Therefore its content has to be structured based on the TRADR ontology and can be queried accordingly.[13] Also, in contrast to the low-level database where speed was the main feature, here consistency is the keyword. As mentioned, in a real scenario, there can be multiple semantic interpreters or other reasoning threads that are writing into the database simultaneously. Therefore consistency must be assured at all times.
To the extent of this work *Stardog* was used as high-level database. "It is a semantic graph database, implemented in Java. It provides support for Resource Description Framework (RDF) and all Web Ontology Language (OWL) profiles providing extensive reasoning capabilities and uses SPARQL (section 4.1.2) as a query language."[17]

Resource Description Framework (RDF)

"The RDF data model is based upon the idea of making statements about resources in the form of subject - predicate - object expressions. These expressions are known as triples in RDF terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, one way to represent the notion "The sky has the color blue" in RDF is as the triple: a subject denoting "the sky", a predicate denoting "has the color", and an object denoting "blue"."[18]
Tables 4.1, 4.2 and 4.3 provide complete triple representations of high-level image- , detection- and victim-objects respectively, as used in this work.

Query language (SPARQL)

SPARQL (pronounced "sparkle") is the standard RDF query language. It is able to access and manipulate data stored in Resource Description Framework (RDF) format. There are three types of queries used by the interpreter. Those are *INSERT*, *SELECT* and *DELETE-INSERT-WHERE* queries. The last one serves as update query.
Listing 1 depicts the SPARQL *SELECT* query used to query for all victims in the database with all corresponding detections, images and attributes. For simplicity reasons example insert and update queries are not depicted here. All syntax definitions can be found under `https://www.w3.org/`

```
PREFIX : <http://www.semanticweb.org/ontologies/tradr#>
SELECT ?vic ?vic_ref ?avgConf ?vic_loc ?vic_loc_ref ?vic_long
        ?vic_lat ?vic_alt ?vic_coo?im ?im_ref ?im_stamp ?im_loc
        ?im_loc_ref ?im_long ?im_lat ?det ?det_ref ?box ?conf ?dt ?fc6
        ?fc7 ?tar_loc ?tar_loc_ref ?tar_long ?tar_lat ?tar_alt ?tar_coo
WHERE{
        ?im a :Image;
        :reference ?im_ref;
        :hasTimestamp ?im_stamp;
        :hasLocation ?im_loc;
        :hasDetection ?det.
        ?im_loc a :Location;
        :latitude ?im_lat;
        :longitude ?im_long.
        ?det a :Detection;
        :reference ?det_ref;
        :fromImage ?im;
        :hasBox ?box;
        :wasDetectedAs "Victim";
        :hasConfidence ?conf;
        :hasDetectionTime ?dt;
        :hasFC6Features ?fc6;
        :hasFC7Features ?fc7;
        :hasTargetObject ?vic;
        :hasTargetLocation ?tar_loc.
        ?tar_loc a :Location;
        :latitude ?tar_lat;
        :longitude ?tar_long;
        :altitude ?tar_alt;
        :mapCoordinates ?tar_coo.
        ?vic a :Victim;
        :reference ?vic_ref;
        :avgConf ?avgConf;
        :depictedIn ?im;
        :hasLocation ?vic_loc.
        ?vic_loc a :Location;
        :latitude ?vic_lat;
        :longitude ?vic_long;
        :altitude ?vic_alt;
        :mapCoordinates ?vic_coo.
}
```

Listing 1 SPARQL *SELECT* query to receive all high-level victim objects in the database with all corresponding images, detections and attributes.

High-Level Image and Location Triples

Subject	Predicate	Object
image-id	a	**Image**
image-id	hasReference	*reference*
image-id	hasTimestamp	*stamp*
image-id	hasLocation	**location-id**
location-id	a	**Location**
location-id	hasReference	*reference*
location-id	hasLatitude	*latitude*
location-id	hasLongitude	*longitude*

Table 4.1 RDF triples representing a high-level image-object with corresponding location-object. High-level objects as predefined by the ontology are highlighted bold and blue. Place holders for the actual data are highlighted italic and green. Note that some predicates might differ slightly from the implementation.

4.2 Low-level data extraction

In the following the low-level side of the semantic interpreter will be illuminated. Included are the object detection, feature vector extraction, as well as the two target localization methods that have been developed in the frame of this work. Although there can be several semantic interpreters for different raw data types and high-level content, in this work however, we are only concerned with victims, that have been detected in normal rgb-images. Thus, in the following, we will pretend there is only one semantic interpreter.
The goal of this process is to collect all required information to create RDF triples according to table 4.1 for images and table 4.2 for detections. Most of this information is required by the high-level reasoning process later on. Therefore, the semantic interpreter is polling the relevant low-level data collection periodically for new images. Each image already contains a time-stamp and GPS-coordinates of the robot's location that are both coined at recording time. In case multiple new items are found in the database, the data collection process described in the following sections will then be executed for each of them. Similarly, localization and feature vector extraction is applied for every detection within the same image. When finished, new triples are inserted into the high-level database and the image is marked as processed. Otherwise, if nothing is detected at all, then there is no need to create detection nor image triples. The image is then marked as processed only.

4.2.1 Detection and feature vector extraction

Chapter 2 provided a brief overview of `fast-rcnn`'s performance results on artificial victim dataset. A "victim" in our case is simply a detection of a person. The pre-trained neural network model used for this work is called `VGG_CNN_M_1024` and can be found here: `https://github.com/rbgirshick/py-faster-rcnn`. It is already trained to classify 20 (21 if you count background) different classes like aeroplanes, bicycles, cars, boats, ..., just to name a few. [2] Therefore, the victim in this work can easily be replaced by a car or barrel or any other high-level objects that are of interest to the rescuers, as long as the corresponding classifier is provided by the network. If not,

High-Level Detection Triples

Subject	Predicate	Object
image-id	hasDetection	**detection-id**
detection-id	a	**Detection**
detection-id	hasReference	*reference*
detection-id	fromImage	**image-id**
detection-id	hasBox	*bounding-box*
detection-id	wasDetectedAs	"Victim"
detection-id	hasConfidence	*confidence*
detection-id	hasDetectionTime	*detection-time*
detection-id	hasFC6Features	*fc6-feature vector*
detection-id	hasFC7Features	*fc7-feature vector*
detection-id	hasTargetObject	**victim-id** or "None"
detection-id	hasTargetLocation	**location-id** or "None"
detection-id	hasOccupationPyramid	**pyramid-id** or "None"
target-location-id	a	**Location**
target-location-id	hasReference	*reference*
target-location-id	hasLatitude	*latitude*
target-location-id	hasLongitude	*longitude*
target-location-id	hasMapCoordinates	*coordinates*
pyramid-id	a	**OccupationPyramid**
pyramid-id	hasOrigin	*origin*
pyramid-id	hasTopLeftDirection	*top-left vector*
pyramid-id	hasTopRightDirection	*top-right vector*
pyramid-id	hasBotLeftDirection	*bot-left vector*
pyramid-id	hasBotRighDirectiont	*bot-right vector*

Table 4.2 RDF triples representing a high-level Detection-object with corresponding Location and occupation-pyramid. Note that the target object, target location and occupation-pyramid can be "None". High-level objects as predefined by the ontology are highlighted bold and blue. Place holders for the actual data are highlighted italic and green. Also note that some predicates might differ slightly from the implementation.

classifiers can also be exchanges or retrained without much effort to fit the rescuers specifications. Moreover, one could think of using even deeper network models in order to improve performance, given the necessary requirement like enough GPU memory.

Detection Parameters

The performance of the neural network and therefore of the semantic interpreter depends on some crucial parameters. Here only the two most important are quickly discussed:

The Detection threshold is most relevant as it decides if a potential victim will be detected or not. Consequently one would like to choose it rather small. On However, this will lead to unwanted detections and multiple detections of a target in the same image. Latter is not that much of a problem as None Maximum Suppression (NMS) filters overlapping detections. Unwanted detection (false-negatives) though, will result

in false victims to be created and thus cause even more miss-assignments of detections. But most importantly, more detections lead to more computational costs due to localization and assignment effort. (More about computational cost and timing can be found in sections 5.2.1 and 5.6).

The Number of Region of Interest (ROI) proposals calculated for each image is the second most relevant parameter. It is the number of boxes within a image, proposed by a RPN, for which the classifiers are applied. The higher this number, the more likely a target will be detected because a certain box might enclose the target more accurately and consequently achieve a higher score. However, there is a limit to improvement as at some point proposed regions are redundant. A compromise has to be found here as detection time increases proportional to this number.

Feature vector extraction

Feature vectors can be seen as the particular pattern of how neurons in the neural network "fire" on a given input (see 2.1.3). For the development of all feature similarity related methods in this work, the assumption has been made that multiple detections of the same target have a overall higher similarity between their feature vectors than detections of different targets.
Moreover, feature vectors extraction in the frame of this work simply means saving the output of the last two layers of the neural network, in case a target was detected. The vector extracted from the last layer called fc7 (fully connected) contains 1024 elements, whereas the vector of the penultimate layer called fc6, contains 4096 elements. "Layer fc6 is fully connected to pool5. To compute features, it multiplies a 4096×9216 weight matrix by the pool5 feature map. Layer fc7 is implemented by multiplying the features computed by fc6 by a 4096×4096 weight matrix."[2]
The idea behind using two different layers for similarity computation and comparison is to take a bigger variety of features into account. This is because each layer can be seen as detector itself reacting on slightly different features.

4.2.2 Target localization

This section explains the derivation of location information for detected targets. This is particularly important as the subsequent reasoning process that is assigning detections to its corresponding victims is mainly based on location comparison.
Let's assume a image was acquired by one of the robots cameras that are used for detection at a certain point in time and at a certain location. Each of the four cameras has its own camera coordinate frame. Let's further assume that the data was uploaded into the main LLDB and is now available to the semantic interpreter.
Two methods have been developed during this work to derive information about a target's location. First uses depth information extracted from a point-cloud in order to find a precise location of the detected target. This algorithm is explained in detail in the following section 4.2.2. However, 3d-points representing the target are not always available at the time of interest. Reasons for that are given in sections 5.2.4 and 5.2.5. Without this information, a exact position can not be derived and thus corresponding detections not assigned to a victim. Consequently the detection would be useless. To prevent that, a second less accurate localization method was developed.
The idea of the seconds method is: Although depth information is missing, by knowing the position of the robot at the time the image was taken, plus the camera frame (direc-

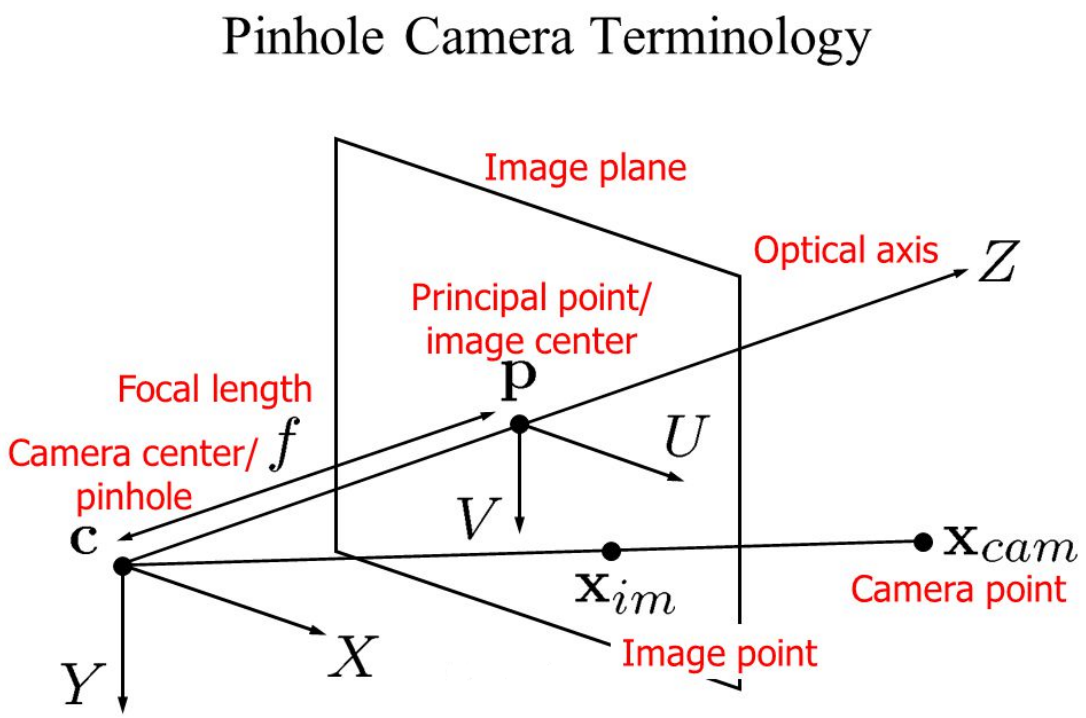


Figure 4.2 Pinhole Camera Terminology as provided by [19]

tion) and also the detection bounding-box, then a 3-dimensional occupation-pyramid can by calculated. It describes an area of space in the global coordinate-frame that is shaped like a tilted pyramid with the camera lens at its peak. But moreover it contains the detected target. Section 4.2.2 is illustrating this method.
Depending on which location information was be derived for a particular detection, the assignment algorithm also has to be adapted. See section 4.3 for detailed explanation.

Position derivation

In order to determine the global position of detected target, depth information is needed to calculate its position relative to the robot first. This is achieved by extracting those 3d-points from the point-cloud that represent the target. For this to be possible, the camera and point-cloud's field of view have to be overlapping. This mostly is the case for four different cameras, as the laser-scanner has a field of view of 270 degree (section3.1).[10]
Moreover, the point-cloud has to be recorded during the time the image was taken. Here discrepancies naturally arise, due to the fact that newly recorded 3d-points are published only every 3 seconds. Consequences on the implementation are discussed in 4.4.3.
Also required are the transformations from the point-cloud frame to the camera frame as well as from the camera frame to the global map frame at image time. Furthermore, calibrated camera information is necessary as it contains the camera matrix and distortion coefficients.

The first step is then to transform the whole point-cloud into camera frame using a

translation vector t and a rotation matrix R. Subsequently, the point-cloud is projected onto the 2-dimensional image plane. Both steps are provided by `OpenCV function: projectPoints` [20]. For a so-called pinhole camera model, this is equivalent to equations 4.1 below. Also compare to figure 4.2. "Real lenses usually have some distortion, mostly radial distortion and slight tangential distortion."[20] Therefore, equations 4.2 - 4.5 are equivalent to equations 4.1 above (for $z \neq 0$), but extended by the distortion coefficients.

$$p' = A\,[R|t]\,P' \tag{4.1a}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.1b}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} + t \tag{4.2}$$

$$x' = x/z \tag{4.3a}$$
$$y' = y/z \tag{4.3b}$$

$$x'' = x'\left(\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6}\right) + 2p_1 x' y' + p_2(r^2 + 2x'^2) \tag{4.4a}$$
$$y'' = y'\left(\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6}\right) + 2p_2 x' y' + p_1(r^2 + 2y'^2) \tag{4.4b}$$
$$r^2 = x'^2 + y'^2 \tag{4.4c}$$

$$u = f_x \times x'' + c_x \tag{4.5a}$$
$$v = f_y \times y'' + c_y \tag{4.5b}$$

where:

- (x, y, z) are the coordinates of a 3D point P in the world coordinate space
- (u, v) are the coordinates of the 2D projection point p in pixels
- (c_x, c_y) is a principal point that is usually at the image center
- (f_x, f_y) are the focal lengths expressed in pixel units
- A is the camera matrix, or a matrix of intrinsic parameters
- $[R|t]$ is the joint rotation-translation matrix
- k_1, k_2, k_3, k_4, k_5, and k_6 are radial distortion coefficients
- P' and p' are the augmented representations of the points P and p respectively

as provided by [20]

Note that in the implementation both steps just described are conducted separately, so that, before projecting all 3d-points to the image plane, all those with $z < 0.5$ are

filtered. In other words: All points that lie behind the camera and those less than half a meter in front of the camera. This operation prevents points to be projected onto the image plane that cannot represent a victim as they do not lie in the field of view of the current camera frame. It also reduces computation effort as point-clouds contain several thousands of points.
Now that we have projected the filtered 3d-points onto the 2d-plane, we can identify those 3d-points, whose 2d-equivalents fall into the detection bounding-box of the target we want to locate. Figure 4.3 depicts two detections with corresponding bounding-boxes in red. Since bounding-boxes never enclose a target perfectly, the set of 3d-points obtained so far still contains "background" points. To make sure only target points are used to derive a position, a simple assumption is made, that is: Target points are most likely found in the middle of the bounding-box. Therefore points are identified that are projected into a smaller bounding-box at the center of the target's bounding-box. These smaller boxes are depicted green in figure 4.3. Their initial box size is set to a tenth the size of the original one. It is then iteratively enlarged in 10% steps of the target's box size until a minimum number (arbitrary threshold - 50 was used here) of 3d-points are identified or the original box size is reached. In the first case a simple outlier filter is applied, that filters points who's root mean square error (RMSE) is smaller than $\lambda\,\sigma$ in any axis:

$$\sqrt{(x-\bar{x})^2} < (\lambda\,\sigma_x)$$
$$\sqrt{(y-\bar{y})^2} < (\lambda\,\sigma_y)$$
$$\sqrt{(z-\bar{z})^2} < (\lambda\,\sigma_z)$$

where (x, y, z) are the coordinates of a 3d-point P, $\bar{P}$ being the mean of all 3d-points, σ the standard deviation and λ a variable threshold. Finally all successful 3d-points are averaged to a single position vector that is then transformed into the global frame. In the case that not enough points are found, the algorithm fails to obtain a location. Furthermore, inaccurate positions can be obtained if the detection is not located in the middle of its bounding-box, as it is the case for detection 2 in figure 4.3. More about that in 5.2.4.

Occupation-pyramid derivation

When depth information is missing, the robots position, the camera frame plus the detections bounding-box can still be combined to derive global location information about the target. The only difference is that without depth, the target cannot be located along the line of sight.The alternative idea presented here is to reduce the cameras field of view (FOV) as depicted in figure 4.4 to a smaller sub-field of view, defined by the detection's bounding-box. This sub-field of view will be referred to as "occupation-pyramid" in this work. Therefore the four corner pixel that describe the bounding-box (compare to figure 4.3) are projected into 3-dimensional space. This is the same projection operation as described in the previous section 4.2.2 but in the opposite direction. Therefore the equations have to be solved for the 3d-point coordinates (x, y, z) as depicted below:

a) Detection 1 - good box **b)** Detection 2 - bad box

Figure 4.3 Example detections with bounding-boxes in red. Filtering boxes are depicted green. Detection 1 serves as positive example, whereas the algorithm fails to filter victim points for detection 2. It illustrates that the target does not always have to be located in the center of the bounding-box.

$$
\begin{aligned}
x' &= (u - c_x) \,/\, f_x \\
y' &= (v - c_y) \,/\, f_y \\
x &= x' \cdot z \\
y &= y' \cdot z \\
&\text{where } z = 1.0
\end{aligned}
$$

where:

- (x, y, z) are the coordinates of a 3D point in the world coordinate space
- (u, v) are the coordinates of the projection point in pixels
- (c_x, c_y) is a principal point that is usually at the image center
- (f_x, f_y) are the focal lengths expressed in pixel units

as provided by [20]

This time however, the distortion model as shown by equations 4.4 is not taken into account for simplicity reasons. Section 5.5 comments on that.
The resulting four vectors or rays then describe the directions from the camera lens towards the four corner pixel in the image plane. Combined with one more vector pointing to the position of the lens, an occupation-pyramid is therefore fully described by a set of 5 vectors. Those can now be transformed from the camera frame into the map frame. Care must be taken here: The four ray vectors are directions and must therefore be transformed by rotation only. In contrast, the position vector of the lens has to be translated as well.
If the occupation-pyramid was derived successfully, the new detection can now be assigned to existing victims by testing whether they are located within the occupation-pyramid. This will be discussed in section 4.3.2.

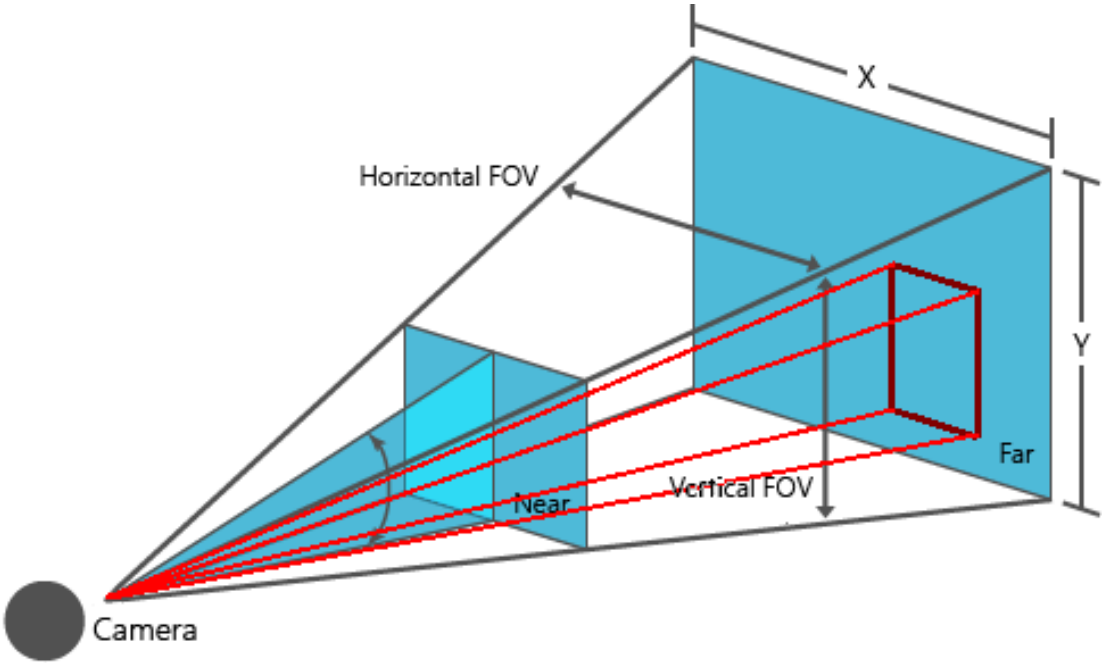


Figure 4.4 Classical camera field of view. Example occupation-pyramid is marked in red.

4.3 High-level victim reasoning

Now that the reader is familiar with the concept of a semantic interpreter, its databases and the low-level data extraction, this section discusses the high-level reasoning process. Its main goal is to create and manage high-level objects as a representation of the extracted data. This must be done in a way that high level consistency is assured.[13] As mentioned, the high level database uses RDF language in order to create a directed graph structure. After the low-level process has gathered all information, it creates high-level detection objects and inserts them into the database. Table 4.1 and 4.2 provide the RDF triple representations of high-level image and detection objects respectively.
The reasoning process begins with querying the database for detections with attribute `hasTargetObject = "None"` . The query result represents new detections. Those must have a `Location` or a `OccupationPyramid` attribute, that determines which of the two assignment processes discussed below will be executed.
Then the database is queried for detections with `targetObject = Victim-id` (compare to SPARQL query 1), returning detections that have already been assigned to a high-level victim. Those are then sorted by `victim-id` into a dictionary structure that represents the temporal working copy of the high-level status. High-level RDF victim triples are shown in table 4.3.

4.3.1 Assignment of detections with position

This section illuminates the core of the reasoning process for detections with available position information (`hasLocation` $\neq$ `"None"`). The goal is to find the victim which is most likely represented by the current detection (best-fit candidate). Subsequently a "yes or no" decision is required that tells whether or not the best-fit victim candidate is actually represented by the new detection. If the result is positive the detection will be assigned to the candidate, otherwise a new victim will be created with the detection. Thus the whole process discussed here has to be executed for every new detection that

High-Level Victim Triples

Subject	Predicate	Object
victim-id	a	**Victim**
victim-id	hasReference	*reference*
victim-id	hasAvgConfidence	*avg. confidence*
victim-id	hasLocation	**location-id**
victim-id	hasDetection	**detection1-id**
victim-id	hasDetection	**detection2-id**
victim-id	depictedIn	**image1-id**
victim-id	depictedIn	**image2-id**

Table 4.3 RDF triples representing a high-level victim-object with corresponding location, detections and images. Note that a victim can have several detections and images. High-level objects as predefined by the ontology are highlighted bold and blue. Place holders for the actual data are highlighted italic and green. Also note that some predicates might differ slightly from the implementation.

has to be assigned.
The idea behind this algorithm is to combine location, as well as feature vector comparison in a way to obtain the best possible result. However, feature vector comparison is rather experimental and was not evaluated at all prior to this work. As a consequence, the developed algorithm is implemented in a way that it can make its decisions only based on location comparison or that similarity comparison is used only if there is sufficient information available to make accurate predictions. In particular that means only if a minimum number (see section 4.3.1) of detections have already been assigned to the victim of interest.

Location comparison

Comparing the locations of a new detection to those of existing victims is quite intuitive. First, the positions of each temporal victim is recalculated as the mean value of all positions of its detections. This is necessary as a new detection might have been assigned to that victim in a previous iteration which would then have changed the average victim position. Second, the distances from the current detection to all existing victims are obtained simply by calculating the 3-dimensional euclidean distance $\mathbf{d}_{eu}$ defined as:

$$\mathbf{d}_{eu}(\mathbf{p},\mathbf{q}) = \mathbf{d}(\mathbf{p},\mathbf{q}) = \sqrt{(\mathbf{p_1},\mathbf{q_1})^2 + (\mathbf{p_2},\mathbf{q_2})^2 + \cdots + (\mathbf{p_n},\mathbf{q_n})^2} = \sum_{i=1}^{n} (\mathbf{p_i},\mathbf{q_i})^2 \quad (4.6)$$

where $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ are vectors and $\mathbf{p}, \mathbf{q} \in \Re^n$.

Those are then sorted by increasing distance. Subsequently a simple threshold filter dismisses all those victims with a distance bigger than a certain threshold to the newly detected target. The threshold can be arbitrarily high but should not be less than 3.0 meters so that the true candidate (if exists) will not be filtered as well.
We now have a set of possible victims whose distances to the new target are smaller than the chosen threshold and which are sorted by increasing distance as well. This set can also be empty which leads to the creation of a new victim.

Feature similarity comparison

The method of comparing feature vector similarities is primarily used to decide whether or not the best-fit victim candidate is represented by the detection that has to be assigned. Furthermore, it can be used to identify a best-fit candidate among a set of possible victims, similar to the location comparison method explained above, however, this time by comparing similarity instead of distance. This secondary usage will only become relevant in section 4.3.2 but is explained here in the context of the equations. Moreover, as a requirement to computing feature similarities, the victims must already have at least three detections assigned to them so that the equations can be applied. In case of the primary usage, this applies only for the candidate that is to be tested. Note that this value is a minimal threshold and can be increased to archive a higher reliability of this method which is especially important for the primary usage. Moreover, by setting it absurdly high (> 100) feature comparison can be deactivated completely which leads to assignment "by distance only".
The comparison is done in the following way: At first, for every possible victim candidate a mean similarity vector is derived which is a measurement of how similar the current detection is to all those assigned to that specific victim in average. Therefore the similarity for both feature vectors (fc6 and fc7, see 4.2.1) between the new detection and every detection of the victim has to be calculated first. Three well known methods are used to compare them namely: The Euclidean Distance, the Manhattan or Taxicap Distance and the Cosine Distance. They are given by equations 4.6, 4.7 and 4.9 and are here denoted as $\mathbf{d}_{eu}$, $\mathbf{d}_{man}$, $\mathbf{d}_{cos}$, respectively. Note that higher similarity results in smaller distances for all three comparison methods because similarity and distance are inverse proportional to each other.

$$\mathbf{d}_{man}(\mathbf{p},\mathbf{q}) = \mathbf{d}_1(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\|_1 = \sum_{i=1}^{n} |\mathbf{p_i},\mathbf{q_i}| \tag{4.7}$$

$$\mathbf{s}_{cos}(\mathbf{p},\mathbf{q}) = \cos\theta = \frac{\mathbf{p}\cdot\mathbf{q}}{\|\mathbf{p}\|\|\mathbf{q}\|} = \frac{\sum_{i=1}^{n}\mathbf{p}_i\mathbf{q}_i}{\sqrt{\sum_{i=1}^{n}\mathbf{p}_i^2}\sqrt{\sum_{i=1}^{n}\mathbf{q}_i^2}} \tag{4.8}$$

$$\mathbf{d}_{cos}(\mathbf{p},\mathbf{q}) = \frac{\arccos\left(\mathbf{s}_{cos}(\mathbf{p},\mathbf{q})\right)}{\pi} \tag{4.9}$$

where $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ and $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ are vectors with $\mathbf{p},\mathbf{q} \in \Re^n$ and $\mathbf{s}_{cos}$ representing the cosine similarity. Consequently two different feature vectors (fc6 and fc7), compared by three different methods, result in a similarity vector $\mathbf{V}_{sim,\,ext}$ of length six:

$$\mathbf{V}_{sim,\,ext} = [\mathbf{d}_{eu,fc6},\ \mathbf{d}_{man,fc6},\ \mathbf{d}_{cos,fc6},\ \mathbf{d}_{eu,fc7},\ \mathbf{d}_{man,fc7},\ \mathbf{d}_{cos,fc7}]$$

Due to the fact that a possible victim must at least contain three detections to be tested by this method, multiple of those similarity vectors are derived for a candidate, one for ever detection. Subsequently the mean of all those similarity vectors is calculated, resulting in only one mean feature similarity vector $\bar{\mathbf{V}}_{sim,\,ext}$ of same length. The index *ext* indicates similarity between a specific victim and a external detection.

$$\bar{\mathbf{V}}_{sim,\,ext} = [\bar{\mathbf{d}}_{eu,fc6},\ \bar{\mathbf{d}}_{man,fc6},\ \bar{\mathbf{d}}_{cos,fc6},\ \bar{\mathbf{d}}_{eu,fc7},\ \bar{\mathbf{d}}_{man,fc7},\ \bar{\mathbf{d}}_{cos,fc7}]$$

We now have derived a vector that describes the similarity between the new detection and a victim candidate. This is done for every possible candidate. To be able to decide, if the similarity to a certain candidate is high enough, we need two things:

First, the mean victim intern similarity which is the mean vector of all similarity vectors obtained by comparing the victim's detections to each other. Second, we need to know how much the intern similarity varies. This corresponds to the mathematical concept of variance and standard deviation. Finally we have all we need to make a decision (primary usage):

The detection belongs to the candidate victim, if the root mean square error (RMSE) between internal mean $\bar{\mathbf{V}}_{sim,\,int}$ *and external mean* $\bar{\mathbf{V}}_{sim,\,ext}$ *lies within a given multiplicity* λ *of the intern standard deviation* $\sigma_{sim,\,int}$*, for at least 5 out of 6 comparison methods, that is entries of the similarity vectors.*

$$\sqrt{(\bar{\mathbf{V}}_{sim,\,int,\,k} - \bar{\mathbf{V}}_{sim,\,ext,\,k})^2} \;<\; (\lambda\,\sigma_{sim,\,int,\,k}),\; for\; k \;\in\; \{1,\,2,\,3,\,4,\,5,\,6\} \tag{4.10}$$

where $\sigma_{sim,\,int} = [\sigma_{eu,fc6},\, \sigma_{man,fc6},\, \sigma_{cos,fc6},\, \sigma_{eu,fc7},\, \sigma_{man,fc7},\, \sigma_{cos,fc7}]$

In the case of the secondary usage, only a single value has to be derived as a candidate score. Based on this score, possible victims can be easily compared and finally a best-fit candidate identified. Therefore, for every entry in the similarity vectors, the mean square error is expressed in terms of a multiplicity λ_k of the corresponding standard deviation. The final score is then simple the mean value:

$$\lambda_k = \frac{\sqrt{\left(\bar{\mathbf{V}}_{sim,\,int,\,k} - \bar{\mathbf{V}}_{sim,\,ext,\,k}\right)^2}}{\sigma_{sim,\,int,\,k}} \tag{4.11a}$$

$$score = \frac{\sum_{k=1}^{6} \lambda_k}{6}\,, \quad for\; k \;\in\; \{1,\,2,\,3,\,4,\,5,\,6\} \tag{4.11b}$$

This method is of relevance in the following section 4.3.2, where a best-fit candidate cannot be derived by location comparison. For the purpose of making a accurate "same victim or not" decision however, it would be wrong to simply judge on the average score as each method compares different properties and both feature vectors (fc6 and fc7) model different properties as well.
Finally, the reason why this method cannot be applied for less than three internal detections of a victim candidate is the following: Comparing only two internal detections results in only a single similarity vector for which no variance and standard deviation can be obtained. Thus no decision can be made.

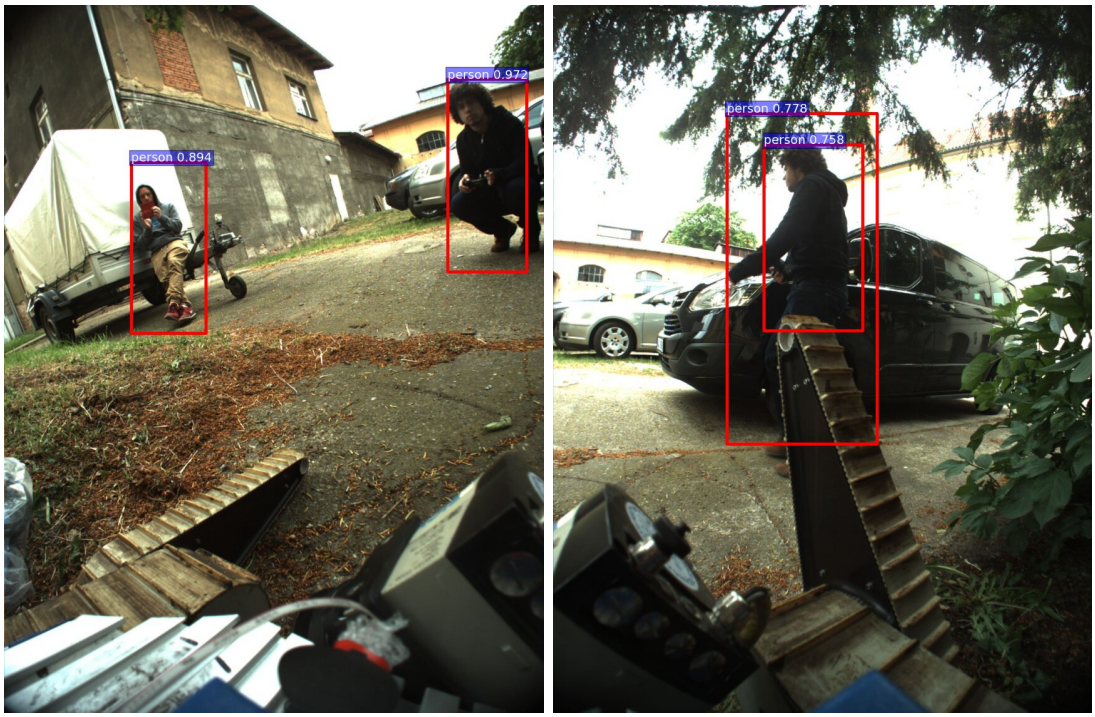
Special case - multiple detections in the same image

There is one special case that should be discussed: Since it its possible that more than one object can be detected in the same image, the easiest way to tell if those represent the same victim is to calculate the overlap $\mathbf{O}$ of their bounding-boxes:

$$\begin{aligned}
&\mathbf{A}_1 = \mathbf{a}_1 \cdot \mathbf{b}_1,\;\; \mathbf{A}_2 = \mathbf{a}_2 \cdot \mathbf{b}_2 \\
&\mathbf{I} = \mathbf{A}_1 \cap \mathbf{A}_2 \\
&\mathbf{U} = \mathbf{A}_1 \cup \mathbf{A}_2 = \mathbf{A}_1 + \mathbf{A}_2 - \mathbf{I} \\
&\mathbf{O} = \frac{\mathbf{I}}{\mathbf{U}}
\end{aligned}$$

where $\mathbf{A}_1$, $\mathbf{A}_2$ are the box areas, $\mathbf{I}$ their intersection and $\mathbf{U}$ their union.
Let's assume two detections have been detected in the same image and one has already

been assigned to a victim while the second one has yet to be tested. Compare to figure 4.6 where detection 6 represents the latter. If the percentage of the bounding-box overlap is more than 50% then it is very likely that both detections depict the same victim. In that case the detection can be safely assigned to this victim right away and no further location or similarity comparison is necessary.
Otherwise, if the percentage of overlap is less than 50%, one can consider the victim which is represented by the first detection as dismissed from the possible candidates, since it cannot be the same victim. This should always be true as long as no miss-assignments have been made earlier. However, if so, then this can lead to consecutive failures. This will be discussed further in chapter 5.4. For now just notice that the dismissal due overlap $< 50\%$ can be disabled in the implementation of this algorithm if needed.



a) Two detections - Overlap $< 50\%$ **b)** Two detections - Overlap $> 50\%$

Figure 4.5 Two detections in the same image. Non overlapping in figure a) which could resemble Image 2 in the assignment scheme 4.6. In contrast, overlapping detections in figure b) could resemble detections 1 & 2 depicted in image 1 (scheme). Detection boxes are marked red.

The decision making

Now that the reader is familiar with the main reasoning building blocks, we can combine them to describe and summarize the decision process for a new detection as a whole:

1. The location comparison is executed as explained in section 4.3.1 above. That leaves us with a non empty set of possible victims, sorted by increasing distance to the newly detected target. A new victim is created in case the set would be empty. In fact, this is always the case, no matter for which reason all the candidates were dismissed.

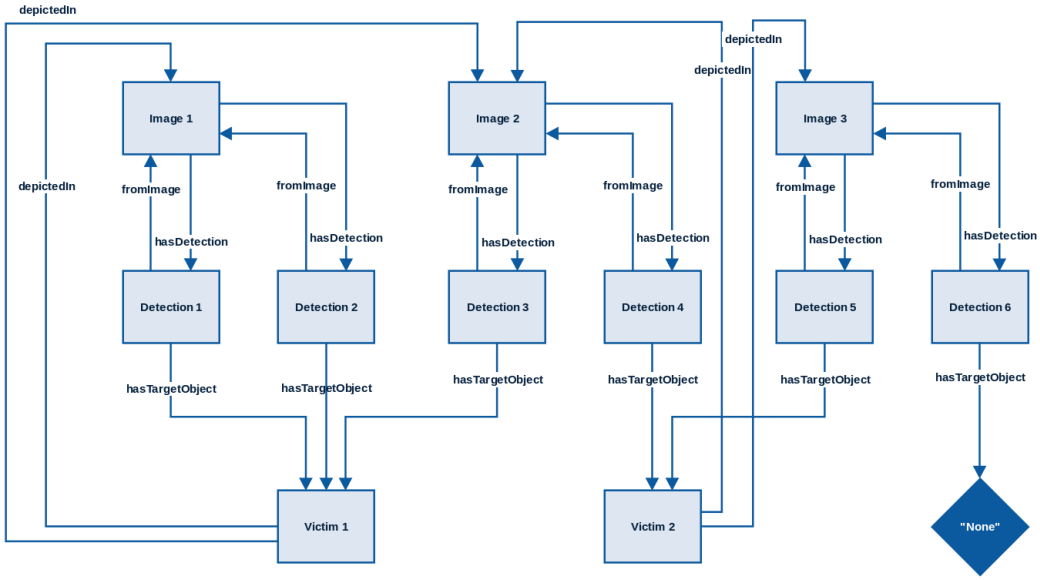


Figure 4.6 Exemplary scheme illustrating the assignment process

2. Feature similarities are calculated. Since the current detection has to be compared to all victim detections anyway, it makes sense to test for the special case (section 4.3.1) here simultaneously. If detections are in the same image and their bounding-box overlap is bigger than 50%, the victim is the represented one, otherwise it will be dismissed (if not disabled).
3. Remaining possible candidates are tested in the previously arranged order of increasing distance to the target by the feature comparison method. The detection is then assigned to the first successful candidate. Again, if non succeeds, a new victim is created.
   As mentioned, a victim candidate must have at least three (or more depending on the threshold) detections, otherwise this method fails. It would then be inequitable to simply dismiss the current candidate and try the next one. Consequently, in this case, the method cannot be used at all to derive a reasonable decision. The algorithm will then switch to "by distance only" mode.
4. If assignment is done "by distance only", the closest candidate is tested whether its distance to the target detection is smaller than a threshold (around 0.5 - 1.5 meter). Here the threshold is equivalent to the radius of a sphere with the new target at its center. If the candidate is not located within this sphere, there is no need to test others as well because it is the closest candidate already. A new victim will then be created.

4.3.2 Assignment of detections with occupation-pyramid

The assignment algorithm for detections that have a occupation-pyramid instead of precise location (`hasLocation` = `"None"`, `hasOccupationPyramid` ≠ `"None"`) is quite similar to the one explained above. However, there are a few differences that need to be outlined:
First and most importantly, these detections can only be assigned to existing victims for which locations already exist. In other words: No new victims can be created in case a detection can not be assigned because the detection cannot provide a precise location

for the new victim. Second, location comparison by distance is not possible. Instead, possible victims are identified if their positions lie within the occupation-pyramid.

Comparing occupation-pyramids with positions

Possible victims for a new detection that only has a target occupation-pyramid are those victims, who's position P_v fall within the 3-dimensional pyramid. The pyramid itself does not have a cut-off value as bottom and is therefore mathematically infinitely high or rather long if it extends horizontally. It is defined by a set of five vectors in total: Four direction vectors ($\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$, $\vec{r}_4$) are describing the pyramid's four edges or rays. They originate from the point P_0 described by position vector $\vec{p}_0$, representing the peak of the pyramid and the position of the camera lens (figure 4.4).

Mathematically, a point (victim position) lies within a pyramid if its distances to each of the pyramid's four faces (planes) all have the same sign.

In general, a 3-dimensional plane consists of all points P with position vector $\vec{p}$, for which the following equation (provided by [21]) holds:

$$\vec{n}_0 \cdot (\vec{p} - \vec{p}_0) = 0 \tag{4.12}$$

when $\vec{p}_0$ is describing a point in the plain and $\vec{n}_0$ being a non zero vector standing orthonormal on the plain and indicating its inclination. The dot here denotes the dot product. This is equivalent to the so called "point-normal form" of a plain (provided by [21]) denoted as:

$$(\vec{p} \cdot \vec{n}_0) + d = 0 \text{ where :} \tag{4.13}$$

$$d = -(\vec{p}_0 \cdot \vec{n}_0) \tag{4.14}$$

We already have a point that lies within all of the four planes needed, that is the peak P_0. Furthermore, the four normal vectors can be derived by taking the cross-product of each pair of adjacent ray vectors:

$$\vec{n}_{0,1} = \vec{r}_1 \times \vec{r}_2$$
$$\vec{n}_{0,2} = \vec{r}_2 \times \vec{r}_3$$
$$\vec{n}_{0,3} = \vec{r}_3 \times \vec{r}_4$$
$$\vec{n}_{0,4} = \vec{r}_4 \times \vec{r}_1$$

Care must be taken here, that all vectors are pointing either to the inside or the outside of the pyramid, since the cross-product is non commutative. The distances d_i are then derived analogue to equation 4.14 for all four planes:

$$d_i = -(\vec{p}_0 \cdot \vec{n}_{0,i}),\ for\ k \in \{1,\ 2,\ 3,\ 4\} \tag{4.15}$$

Finally the four shortest distances D_i of a candidate's position P_v to the pyramids faces can be obtained by:

$$D_i = (\vec{p}_v \cdot \vec{n}_{0,i}) + d_i \tag{4.16a}$$

$$D_i = \frac{|(\vec{p}_v \cdot \vec{n}_{0,i}) + d_i|}{\sqrt{\vec{n}_{0,i} \cdot \vec{n}_{0,i}}},\ for\ k \in \{1,\ 2,\ 3,\ 4\} \tag{4.16b}$$

Where equation b) is the original one provided by [21]. Since we are only interested in the sign rather than the magnitude, the absolute value of the nominator and the normalization become unnecessary. This then results in simplified the equations a).

The decision making

Now we can summarize the decision process for detections with occupation-pyramid instead of a precise location:

1. In contrast to before (4.3.1), here the possible victims are identified by testing if their locations fall into the new detection's occupation-pyramid. Note that ideally here only one or at least less victims should pass this selection than by comparing location distances. This is because, for two victims to be possible at the same time, they must be in the same line of sight from the robots perspective, whereas, by comparing distances, possible victims lie within a sphere with a threshold radius. It can not be generalized though, as it also highly depends on chosen thresholds and the victims' positions relative to each other.
2. If there is more than one candidate, a best-fit victim has to be identified. Since we cannot compare locations as before to determine a candidate order, here the similarity scores are used as calculated by equations 4.11. As mentioned, this method required more than three detections of the candidates. If this requirement is not met, the detection cannot be assigned now, but maybe at a later point in time, when the possible victims are represented by more detections.
3. Finally the best-fit candidate, if exists, is tested by the similarity method's primary usage. If the candidate does fullfil equation 4.10 for at least 5 out of 6 elements in the similarity vectors, then the candidate is successful.

4.4 The simulation pipeline

In the previous sections we have illustrated the interpreter's core functionality and reasoning methodology without integrating it in or respecting its software environment. The final interpreter is supposed to run on as a ROS-node on TRADR's main computer which has good computing resources and combines all the low-level databases acquired by the robots into one. It can be assumed that all data required (see next section) by the reasoning process will then be available instantly. Thus, the interpreter in the final scenario does not have any timing constraints on how fast to process a image or its detections in order to catch up with incoming data. However, project TRADR is still in its development stage and those features are not implemented yet. Therefore a pipeline was developed prior to the interpreter's core functionality to make the simulation possible. The pipeline is playing ROS-bags (see 3.2) to simulate a sortie and to fill the LLDB with images. On top of that the semantic interpreter was developed on a private computer using GPU acceleration for detection. Its implementation as well as arising constraints will be discussed in the following.

4.4.1 Requirements of the reasoning process

Three points are important for simulating the semantic interpreter:

1. The low-level database has to be filled with ROS-messages of type `common_db_msgs/AnnotatedPicture` which contains information about the camera frame, time stamp, location and orientation of the robot when the image was taken.
2. Point-cloud data has to be available for the time the image was taken which leads to problems as the new point-cloud message, containing the new points, in addition to all previously recorded, is only published every three seconds, whereas images are published several times a second.

3. Transformations have to be available from laser to camera frame and from camera frame to the global map frame for the time the image was taken.
4. Camera calibration information is required that contains the camera matrix as well as its distortion coefficients.

4.4.2 Filling the low-level datbase

In order to fill the database in a simulation scenario with image data in form of ROS-message of type `AnnotatedPicture`, a simple ROS-node was implemented. Chapter 3.2 gives a brief introduction to ROS and its features.
The node operates by subscribing to the four image topics corresponding to the four different cameras of the robot which are used for detection. This data comes in the form of ROS-message type `sensor_msgs/Image` which contains the image, time-stamp and camera-frame. However, it is missing location information about where the image was taken. Therefore the node also subscribes to a GPS-topic that provides the current position of the robot. It then puts all information together into a `AnnotatedPicture` message and publishes it under a new topic.
The node also provides the crucial functionality of controlling the data-flow of the simulation. Since the picture rates of the cameras are much higher compared to what the interpreter can process in a certain amount of time, a method is needed to regulate the rate and amount of pictures saved to the database. This is achieved by buffering the `AnnotatedPicture` messages at first. The four buffers, one for every camera frame, are then handled at a given rate f, and then only every i^{th} picture is published until the buffer is empty again.
Subsequently a second simple node (which was already implemented) subscribes to this topic and saves the images to the database under a generated image-id in the form of an UUID. From this point on the data is available to the interpreter.

4.4.3 Point-clouds

Having a point-cloud available for the time the image was taken is crucial for obtaining the locations of newly detected targets. Without a location, a detections can only be assigned by occupation-pyramids and this helps only if the same target object was detected and localized successfully before.
Point-cloud with new points are only published every three seconds whereas images several times a second. The point-clouds time stamp marks the beginning of this three second period. That means the best possible point-cloud for the localization task is the one with a time-stamp difference: $0.0 < (t_{image} - t_{pc}) < 3.0$ with $(t_{image} - t_{pc})$ in seconds. That means the best point-cloud was recorded while the image was taken also. However if that particular point-cloud is just not published yet, the localization process has to wait for it, at the maximum, for the full three seconds if the image was taken right after the last point-cloud was published.
The localization process is also buffering the last k point-clouds in case the the semantic interpreter is lagging behind so that it does not have to wait then as well. It should be mentioned again that a new cloud message always contains all the points recorded since the beginning of the sortie.

4.4.4 Transformations

Getting the required transformation is at least evenly important. Without them, neither target positions, nor target occupation-pyramids can be calculated. In the ROS-environment, transformations can be buffered up to a desired duration as well. In case however, that the interpreter is lagging more than the buffer's duration behind, transformations can become unavailable. Then the algorithm asks for later transformations until it catches up with the last entry of the buffer. But in fact there is only one transformation that should be used and that is stamped exactly at image time. Every other transformation would result in erroneous calculations, unless the robot was not moving.

5 Test results and evaluation

The purpose of this chapter is to analyse the performance of the semantic interpreter and outline its weaknesses. This is done by evaluating the different methods and building blocks of the interpreter that have been developed during this work, separately. Prior to that, tow main simulation scenarios that have been used for testing, are described. Note that the main goal of the overall work was to implement the interpreter's main functionality together with the necessary simulation pipeline. Thus no sophisticated evaluation methods have been implemented here. This means that the following analysis is based on empirical experience acquired throughout iterative testing. However the following statements will be reasonable justified.

5.1 Simulation scenarios

As explained in 4.4, the semantic interpreter is simulated by playing ROS-bags which are recordings of a robot sortie. Since no data of real disaster events exist, some "artificial" disaster ROS-bags have been recorded that provide all necessary features for simulation and test cases for evaluation. Necessary features are basically the following:

1. The scene of the recording should contain at least tow persons which can be detected as "victims" so that the interpreter has to distinguish between them. Optimally those persons should not change position.
2. The mapping process has to be running, optimally during the recording or while replaying the bag for simulation, so that transformations and positions can be obtained.
3. The laser and the omni-camera have to calibrated and and publishing data.

Two ROS-bag scenarios have been recorded that were mainly used for evaluation. In the following, those will be illustrated and their assignment results presented.

5.1.1 Scenario 1

In this scenario two targets can be detected that are stationary for most of the recording time. Only one changes position during the first and last seconds of the recording. The targets are located roughly 3 meters apart from each other and initially less than 4 meters away from the robot. This distance is decreasing as the robot moves slowly towards them. The resulting victims are depicted in figures 5.1 and 5.2.
Moreover, the robot in this scenario acquired images during the recording that depict both targets at the same time. This provides a useful test-case to evaluate the corresponding assignment mode (explained in 4.3.1, evaluated in 5.4) where decisions are derived, based on the targets bounding-box overlap.
Another feature of this scenario is that one target is detected in two different camera frames (victim 3 a) and b) in figure 5.2) and is still successfully identified as the same victim. This verifies the localization process with respect to conducted point-cloud transformations.
It is worth mentioning that the robot does not move more than 2 meters but is climbing

down a 30 centimetre step and therefore resides in tilted positions during data acquisition (compare angles in figure 5.1 c) and d)). Furthermore, for the purpose of testing the occupation-pyramid assignment in this simulation, precise localization was skipped for every third detection and instead an occupation-pyramid was derived although all required information was available.
Overall, this simulation scenario serves as positive example of the semantic interpreter's performance. Out of 27 detections 25 were assigned correctly whereas only 2 detections without location could not be assigned as no victim fell into their occupation-pyramid (see 5.5). However, the fact that 7 detections out of 9 have been correctly assigned by occupation-pyramid verifies that at least the true candidate was not filtered and that this method (4.3.2) works in principle. Moreover, the primary usage of the feature similarity comparison method (4.3.1) also had to work at least 7 times correctly in this simulation. Furthermore, comparing the position of Victim 1 in figures 5.1 a) and b) to the position of victim 3 in figure 5.1 gives a good demonstration of the localization precision. The stated results are derived by simulating bag `ugv_2016-05-17-11-25-49` with the following main parameters:

- `Detection confidence threshold: 0.75` (4.2.1)
- `Number of region proposals: 350` (4.2.1)
- `Location comparison - "by distance" threshold: 1.2 meter` (4.3.1)
- `Location comparison - filter distance: 3.0 meter` (4.3.1)
- `Similarity comparison - minimal detections: 7` (4.3.1)
- `Similarity comparison - lambda thresh: 2.0` (4.3.1)
- `Same image overlap threshold: 0.5` (4.3.1)
- `Minimal 3d-points: 50` (4.2.2)
- `Pipeline publish rate: 0.25 Hz` (4.4.2)
- `Pipeline image rate: every 10th per camera` (4.4.2)

5.1.2 Scenario 2

Compared to the first one, this recording serves rather as a worst case scenario. Here multiple targets are passing the robot while the robot itself is moving and turning rapidly. Moreover, targets are located further away up to circa 10 meters. The assignment still works fairly good, however some weaknesses of the assignment process and the pipeline could be identified that are treated in the following sections.
The here stated results are derived by simulating bag `ugv_2016-05-17-12-15-54` with the same parameters as in the previous scenario. Notice that results for this recording can vary for different test runs although all configuration parameters are the same. This happens due to small timing fluctuations in the pipeline. The ROS-nodes filling the database is then selecting different images from its buffers that are fed to the interpreter. Consequently different detections are obtained and the result changes as well. Furthermore, here occupation-pyramids are not calculated on purpose for every third detection in contrast to the scenario above but still if the precise localization method fails.
In the conducted test run, 7 out of 9 detections were successfully assigned. 2 detections were miss-treated as a result of inaccurate localization. Figure 5.3 depicts 4 detections of 3 victims that should actually be the same. The reason for the assignment failure is that the targets c) and d) are not located in the middle of their bounding-boxes, nor are they enclosed tightly. However, the three detection of victim 1 demonstrate that

localization can be accurate although the robot drove roughly 5 meters while passing the target.

5.2 Victim localization

Under ideal conditions, the victim localization algorithm developed in this work, is able to derive very precise position estimations. Errors smaller than 0.5 meter have been achieved repeatedly. However there are a few factors that can have a strong influence on the localization algorithm and consequently reduce accuracy.

5.2.1 Timing constraints

It has been stated before that timing is a crucial issue for the interpreter in its current state. When pictures are saved to the LLDB at higher rates than the interpreter can process in real time, then the discrepancy by which the interpreter is lagging behind increases slowly until it exceeds the buffer length for transformations or point-cloud data. It is trivial to see that localization is then erroneous. Therefore, the picture rate was to decreased for simulation purposes. During this work the upper image-rate for which localization was still successful was: Every tenth image of each of the four cameras while publishing and thus saving them to the low-level database every four seconds. For optimal results only every twentieth image was saved. This is discussed in more detail in timing section 5.6.
However, once all point-clouds and transformations are saved to the LLDB as well and therefore available at all times, this will not be of relevance any more.

5.2.2 Odom drift

For now the global location of the robot in the map frame is obtained by integrating accelerations measured by a inertial measurement unit (IMU) on the robot. A well-known problem of IMU's is that they are drifting over time and therefore create localization errors.[22] TRADR plans to design a process to update the IMU data and reset this error, however, not during this work. Once that feature is integrated in the framework, it might be necessary to recompute the whole HLDB content as transformations could change retroactively.

5.2.3 Movements during point-cloud acquisition

It takes about 3 seconds until newly recorded 3d-points are published in a new point-cloud message. Taking an image only a few milliseconds. If the robot is not moving at all during acquisition time, no problems occur. Otherwise, if it is moving and especially rotating then, it is possible that "wrong" 3d-points are projected into the detection bounding-box and therefore the localization will be erroneous. This is still true although a laser-assembler node is looking up the corresponding transformations to the global frame many times during those 3 seconds of acquisition time so that movements are considered in that calculations. However, transformations are derived from the IMU which contains drift errors as explained above. Those errors are therefore also reflected in the in the transformations and thus in the point-cloud.
Still, this is not so much of a problem as the TRADR's ground vehicle move rather slow. Even though some detection might by unusable, if the robot stops for just a

short period of time, then it is likely that sufficient accurate data was acquired so that localization will be successful, at least once. Furthermore, operators can be trained to control the robot smoothly so that this problem is minimized.

5.2.4 Filtering victim points

When the localization process explained in section 4.2.2 filters for 3d-points that represent the target out of all those points projected into the bounding-box, it assumes that the target is most likely located in the middle of it while background occurs more towards the edges of the box. This algorithm was illustrated by figure 4.3.
In the results of test scenario 2 two example detections are presented (see figure 5.3 c) and d)) for which this assumption does not apply. Here the targets are located slightly of the middle and the bounding-box does not enclose it tightly. Consequently more background than target points are filtered and thus, the outlier filter removes true target points. The final position then rather resembles the background than the target.
Alternatively one could use detection algorithms that segment resulting detections pixelwise providing a detection mask instead of a bounding-box. This would lead to much higher accuracy as background outliers become less likely. Also, if this kind of segmentation is not provided by the detection algorithm, state of the art image processing segmentation algorithms could be used to identify 3d-target-point more precisely.

5.2.5 Point-cloud coverage

Section 3.1 provided information about the robots sensors and especially the cameras' and laser's field of view. It is easy to see that the field of view of camera 3 and 5, as depicted in figure 3.1, are not completely covered by the laser's field of view and therefore the point-cloud. Thus, it can happen that no 3d-point can be found that represent the target as non exist. The target was simply not in the field of view of the laser as well. However, it has been sufficiently explained that then, a occupation-pyramid is derived instead to be able to assign the detection anyway. See section 5.5.

5.3 Similarity comparison

Let's assume we have a set of possible victims for a new detection that has to be assigned, then there are two scenarios where feature-similarity comparison can be applied:

1. If a candidate order was already derived e.g. by location comparison, then similarity comparison can be used to decide whether or not the best candidate is represented by the detection in case the candidate contains enough detections already. Enough here means more than three or the minimal threshold. In general, this method was developed as an alternative and more sophisticated way of deriving such decisions in contrast to simply decide upon a simple statical similarity threshold.
2. If no candidate order has been derived yet then similarity comparison can be applied for that purpose in case all candidates have at least three detections. Although this method has been developed for the previous purpose, it is more reliable for deriving a candidate order than to decide one the best-fit candidate. This is because two detections of the same target can have small similarity due to changes in perspective or in the background. But for two detections of different targets (let's say target 1 and 2), it is unlikely that a third detection also representing target 1, has a higher similarity to target 2.

Additionally, since similarity comparison is based on variance and standard deviation which is a statistical concept, accuracy will increase the more empirical data is available. In our case that means the more detections are already assigned to possible candidates. This applies to both methods described above. For the same reasons however, similarity comparison can not be used independent of other methods like location comparison. It only serves as an additional decision maker.
As a more sophisticated comparison algorithm, one could also think of training the network itself to compare its own detection output in order to assign them. This could provide a much higher reliability than the methods derived here.

5.4 Special case

In section 4.3.1 we saw how in the special case, that both detections are in the same image, the assignment process can be abbreviated: If the overlap of both detections is bigger than the threshold, the victims is identified sufficiently.
In the other case however when overlap is smaller that the threshold, it is assumed that it cannot be the same victim and will thus be dismissed from the possible candidates. Here problems can occur. For illustration, let's assume two target have been detected in the same image and their bounding-boxes are not overlapping more than the threshold (usually 50%). Let's further assume both victims have been detected before and therefore already exist in the HLDB: Now if the assignment process starts and incorrectly assigns the first new detection to the other victim in the image, then the second detection can not be assigned to the same and this time correct victim any more. This is exactly because the assignment algorithm finds that this victim already contains a detection in the same image and the overlap is smaller the threshold. It will therefore be excluded from possible candidates.
This will repeat for every upcoming image in which both victims have been detected and consequently lead to more miss-assignments. In addition, similarity comparison will decrease in prediction reliability as well.
So why dismiss victims at all if the overlap is smaller than the 50% threshold? The main argument is that it is still a strong reasoning criterion that works perfectly as long as that initial miss-assignment does not occur, which is a rather rare phenomena. Also for victims to be detected in the same image, they must be located closely together. That in turn implicates a higher probability of miss-assignment due to location inaccuracy. It also implicates the same for similarity comparison as victims in the same environment might share a higher similarity as well, due to similar background. However, this feature can be deactivated easily in the launch file of the semantic interpreter if necessary.

5.5 Occupation-pyramid

The concept of occupation-pyramids was developed during this work as an alternative and backup to the original localization method which can fail for various reasons discussed before. Due to the fact that detections with pyramid can only be assigned to existing victims, this method serves only as a secondary process.
Furthermore, in 4.2.2 it has been stated that the distortion model was neglected for simplicity reasons. Using the distortion as well when projecting 2d-pixels to 3d-space would require equations 4.4a and 4.4b to be solved for x' and y' respectively, which are differential equations of second order. Moreover both equations depend on r (equation

4.4c) which in turn depends on x' and y', which we are trying to derive. Consequently, the decision has been made to stick with the simpler version as it works satisfactorily for the purpose of this work. However, there are a few advantages outlined in the following:

1. Being able to successfully assign those detection for which localization failed as well, will provide more data, based on which the similarity comparison method can derive more reliable scores and decisions.
2. The method only depends on transformations and not on depth information.
3. Deriving occupation-pyramids and checking if a candidate lies within it is a simple task that does not need much computational effort compared to the primary localization method where big point-clouds are handled and filtered leading to many iterations.
4. In section 4.2.2 and figure 4.3 b) we saw how precise localization can fail due to sloppy bounding-boxes. The occupation-pyramid method is immune to this kind of problem, therefore it could be used as a double check or verification mechanism in the assignment process so that both methods would cooperate more effectively and annulling each others weaknesses.
5. Ultimately, one could think of a assignment process that calculates 3-dimensional intersections of occupation-pyramids to derive smaller and more precise habitation zones. Localization would then be completely independent of depth information, however, computing such intersection is not trivial and can be computational much more expensive.

5.6 Timing anlalysis

In the following, timing results for the computational most expensive and therefore most time consuming building blocks of the semantic interpreter are briefly presented and timing related issues are discussed.
With regard to the all timing statements made in this section, notice that the simulations were conducted using the parallel computing platform `CUDA` for GPU (GeForce GT 730M with 2048 MB memory) acceleration and a intel-core i5-4200U processor.
Overall, the semantic interpreter in its current state and using above resources, is capable of processing every tenth image of each of the for camera frames when saved to the low-level database every four seconds in real time. However, if no stronger platform is used, it is advised to only process every twentieth image in order to obtain optimal results. For the following statements, this parameter was used as well. The cameras itself capture images at 15 FPS, or raw uncompressed images at just under 7 FPS.[9] During this work uncompressed images were used.

5.6.1 Detection

Detection is the most expensive part of the interpreter as it is executed for every new image saved to the database. All subsequent task are only required for detections. Although there can be multiple detections in the same image, the a amount of detections is usually still small compared to the amount of images.
Detection for one image takes in average about 0.85 second per image when 350 region proposals are calculated by the RPN. That means if 4 images (one from every camera frame) are saved to the low-level database every 4 seconds than the interpreter is already busy without any further reasoning.

5.6.2 Localization

The main localization method which is executed for every detection can take up to roughly 4.5 seconds. This however does usually not happen multiple times in a row because at most 3.0 seconds of that time the localization thread is waiting for the upcoming point-cloud message. Next time localization is executed, there is no need to wait any more as the whole pipeline is already lagging those 3.0 seconds behind. While waiting, the thread is also being suspended between subsequent point-cloud availability checks so that other threads are not blocked. However, if waiting time is zero, the localization precess can finished within 0.4 seconds.

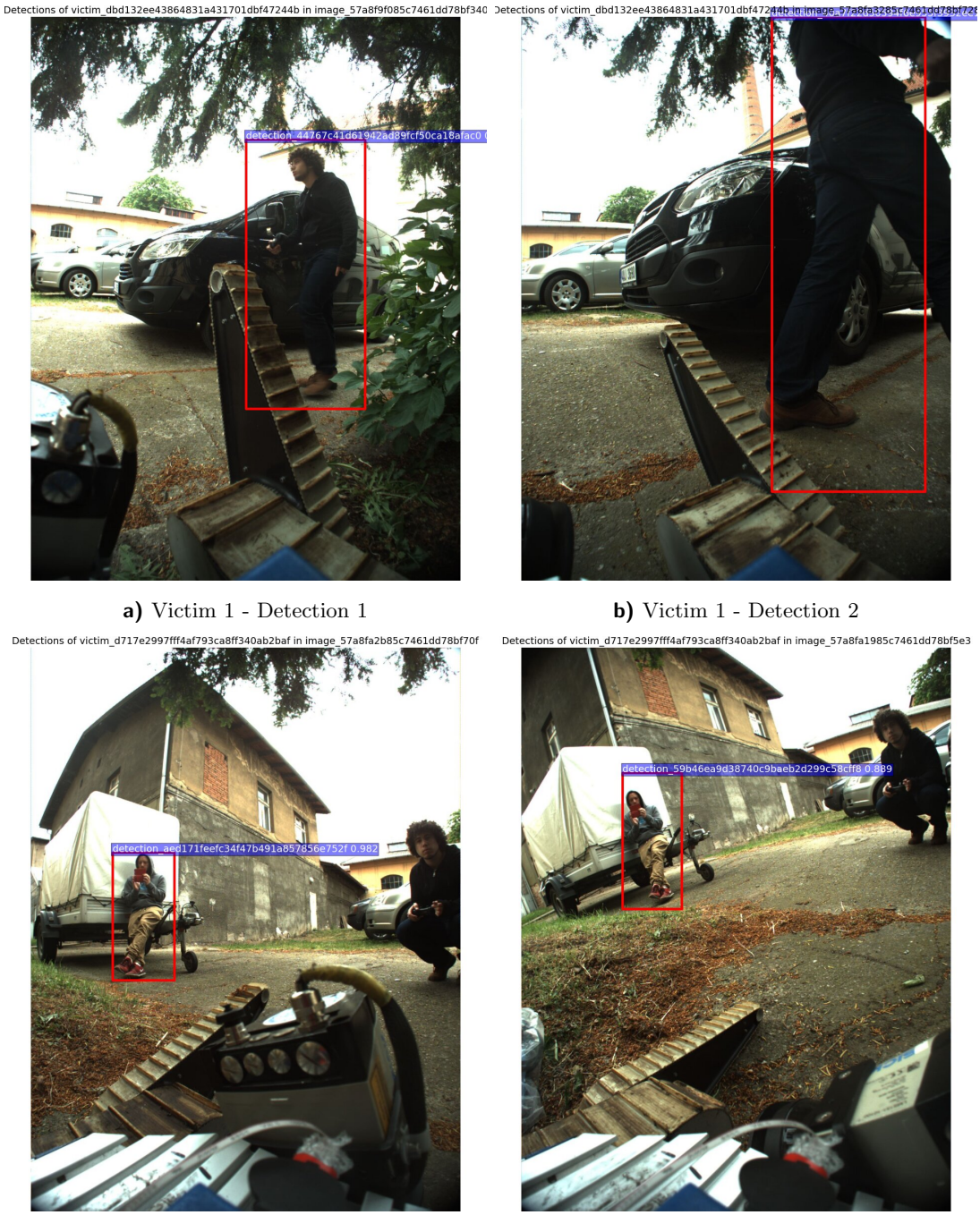
Besides waiting, the two main point-cloud filters together consume second most time up to roughly 1.5 seconds. This varies depending on the point-cloud itself and the field of view for which the cloud is filtered. The first filter discards all 3d-points behind the camera and the second one filters all those that are not projected into the bounding-box. 1.5 seconds still seem surprisingly long. Here might be room for improvement as of the python library `numpy` was not probably used to its full potential.
Furthermore, notice that the point-cloud messages published under `/dynamic_point_cloud` are getting bigger with every new cloud acquired. This is because it contains all points that has been recorded so far all along the way, instead of only those points recorded in the last 3.0 seconds.
Also, computation time of the target filter varies because it iterates up to 10 times over the amount of 3d-points that have been projected into the bounding-box, however, usually this iteration breaks much earlier when enough 3d-points have been identified successfully and most points have been filtered before as well.

5.6.3 Extraction

There is one issue that should be mentioned here: When images are extracted from the LLDB, they are not processed in the same order by which they have been acquired. That means if a image is processed much later as it has been saved the corresponding transformations and messages for that point in time are not buffered any more. The current low-level application programming interface (API) does not provide a method to fetch all new element at the same time or at least their stamps so that they could be sorted prior to processing them. But, once again, this issue is will not be of relevance any more once all information required by the reasoning process are saved to the low-level database and real time constraints dissolve.

5.6.4 Reasoning

The high-level assignment process is also time relevant. Exemplary, for 12 detection with locations and 7 detections without, but occupation pyramid instead, both assignment processes (one full update of the HLDB) took around 12 seconds. Hereby, feature similarity comparison is the most expensive task due to their length of 1024 and 4096 elements for fc7 and fc6 respectively.



a) Victim 1 - Detection 1

b) Victim 1 - Detection 2

c) Victim 2 - Detection 1

d) Victim 2 - Detection 2

Figure 5.1 Resulting "victims" for test scenario 1 part 1 (see 5.1.1). Figure a) and b) depict Victim 1 whereas c) and d) victim 2. Detection boxes are marked red. More results (part 2) for the same scenario are depicted in figure 5.2

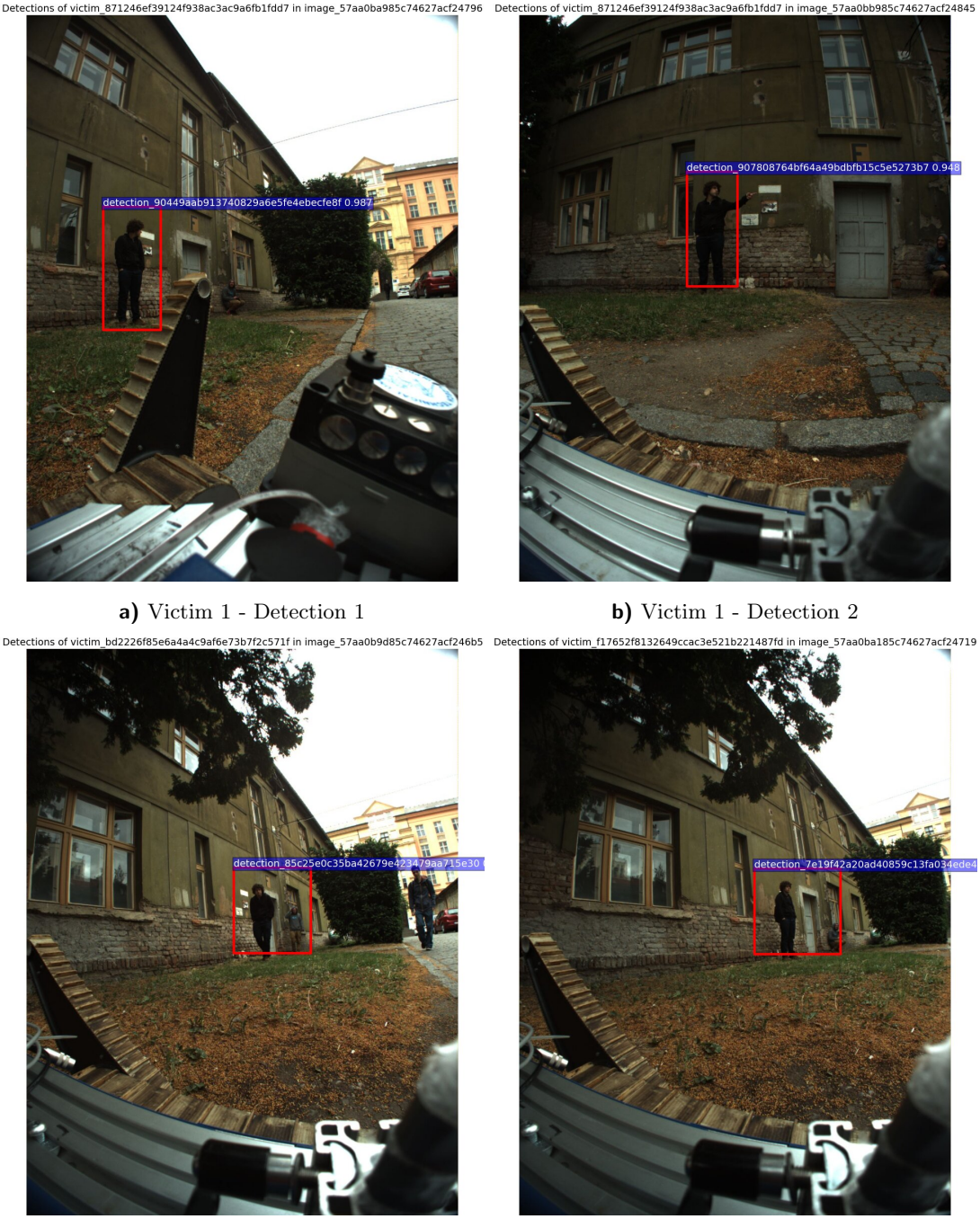
a) Victim 3 - Detection 1

b) Victim 3 - Detection 2

c) Victim 3 - Detection 3

d) Victim 4 - Detection 1

Figure 5.2 Resulting "victims" for test scenario 1 part 2 (see 5.1.1). Figure a), b) and c) depict Victim 3 whereas d) shows victim 4. Notice that Victim 3 in a) is detected in a different camera frames than in b). Detection boxes are marked red.



a) Victim 1 - Detection 1

b) Victim 1 - Detection 2

c) Victim 2 - Detection 1

d) Victim 3 - Detection 1

Figure 5.3 Resulting "victims" for test scenario 2 (see 5.1.2) Figure a) and b) depict Victim 1 whereas c) and d) show separate victims as assignment failure example. Detection boxes are marked red.

6 User manual

This chapter aims to provide all information and links required to install and launch the semantic interpreter. To avoid confusion, notice that the semantic interpreter in the implementation is called semantic modeler instead. The code is written in python.

6.1 Software requirements

The following software packages are required by the semantic modeler:

- The full desktop version of ROS-indigo or newer version has to be installed preferable on ubuntu. The installation should be done according to ros.org.
- `Faster-rcnn` must be installed on the system. Full installation instructions can be found here including `Caffe` and `py-caffe`. Moreover, download links for the pre-trained network models are also provided there. After successful installation, don't forget to export python-path to your `faster` directories and `CUDA` libraries if used. Type the following commands into a terminal or append them at the end of your local `/home/.bashrc` file to make changes permanently.

```
$ export PYTHONPATH="${PYTHONPATH}:/path-to-faster/py-faster-rcnn/lib/"
$ export PYTHONPATH="${PYTHONPATH}:/path-to-faster/py-faster-rcnn
/caffe-fast-rcnn/python"
$ export PATH="/usr/local/cuda/bin:$PATH"
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

- GPU acceleration is not absolutely necessary but strongly advised as it will increase object detection speed by a factor of 10. Moreover the interpreter was never tested for CPU usage only. `CUDA` download link and installation instructions can be found here.

6.2 TRADR installation and build

A basic installation guide can be found in the ciirc-redmine pages. The source code can be downloaded from gitlab. Notice that a account is required for accessing `gitlab` and `redmine` web-pages. Therefore please contact my supervisor Tomáš Svoboda (svoboda@cmp.felk.cvut.cz). In order to set up the databases properly please have a look at the database qiuckstart quide and the launch database guide.
Not all the TRADR's software components are required by the semantic interpreter. As a minimal build only the branches listed below should be cloned from git. For compilation `catkin_tools` are recommended which can be found here. If some packages are complaining during the compilation type:

```
$ roscd [package-name]
$ touch CATKIN_IGNORE
```

Branches:
- `tradr-database [branch modeler_dev]`
- `tradr-msgs [branch master]`
- `tradr-user-interaction [branch master]`
- `tradr-ugv-base [branch master]`
- `tradr-payload [branch tradr_cloud_services]`
- `tradr-utils [branch master]`
- `tradr-vision [branch master]`
- `tradr-loc-map-nav [branch master]`
- `librover [branch master]`

6.3 Usage

Once everything is set up properly, the semantic interpreter can be easily launched using ROS-launch-files. All parameters of the interpreter and the pipeline are included and commented accordingly. Have a look at the provided launch files and make sure the paths to the `caffe network model`, the corresponding `prototxt-file` and the `rosbag-file` are set accordingly.

Start the databases by typing the following command into a terminal:

```
$ rosrun tradr_docker_images start_db.sh [mission name]
```

To start the semantic interpreter separate from the pipeline (advised) enter the following command:

```
$ roslaunch semantic_modeler semantic_modeler.launch
```

Then open another terminal and enter one of the commands below. Both will start the pipeline, however `play_bag_and_fill_lldb_adv.launch` provides a few more options.

```
$ roslaunch semantic_modeler play_bag_and_fill_lldb.launch
$ roslaunch semantic_modeler play_bag_and_fill_lldb_adv.launch
```

To launch the interpreter and the pipeline all at once just enter:

```
$ roslaunch semantic_modeler semantic_modeler_plus_bag.launch
```

7 Summary and Conclusion

In the here presented work, a semantic interpreter was developed. Its purpose is to assist human rescue teams, within the framework of the TRADR research project, to detect, localize and identify victims of disastrous events so that rescue missions can be easily planed based on the extracted and restructured information. Therefore the state of the art machine-learning algorithm `faster-rcnn` was used for victim detection and recognition. In addition, two localization methods have been developed so that multiple detections can be compared as a basis for the subsequent reasoning process. On top of that a feature similarity comparison method has been designed which can be used to derive decisions about victim candidates in two different ways: Either to decide if a best-fit victim candidate is represented by a certain detection or to identify a best-fit candidate among multiple candidates. Furthermore, a high-level reasoning process was developed, that assigns detections to victims and creates new ones. In general, it manages the high-level database in a way that data consistency is assured and information can be queried accordingly. A pipeline was also implemented based on ROS, so that different recorded scenarios can be simulated. Finally every aspect of the semantic interpreter was evaluated and discussed in detail.

It can be concluded that the original goal was accomplished and on top of that methodology has been developed to lift the performance of reasoning process to a satisfactory extent. Many problems that occurred during the development with regard to the simulation pipeline, timing constraints in particular, will become irrelevant once the interpreter can collect all required information from the low-level database directly. However, there is still a lot of room for improvement.
A segmentation algorithm could be used that identifies the target more accurately within its bounding-box in order to extract 3d-target points from the point-cloud. Also occupation-pyramid intersections could be derived as an alternative localization method that would allow complete independence of depth information. Finally the object detection can be enhanced in many ways: Classifiers could be retrained on real disaster datasets that vary from usual data. Deeper neural networks could be used that make more accurate predictions, and eventually the network could be trained to compare its own detections to perfect the assignment process. Thus, the semantic interpreter in its current state is not meant to be the final version but instead, it serves as a strong foundation and template, based on which more sophisticated versions can be derived.

Bibliography

[1] Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim Greeff, Koen Hindriks, Mark Neerincx, Petter Ögren, Tomáš Svoboda, and Rainer Worst. Tradr project: Long-term human-robot teaming for robot assisted disaster response. *KI - Künstliche Intelligenz*, 29(2):193–201, 2015. 5, 11

[2] Ross Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015. 6, 8, 9, 17, 19

[3] Eric Derner. Visual localization and place recognition. Master's thesis, Czech Technical University, Faculty of Electrical Engineering, Department of Computer Science, 2015. 6, 7

[4] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011. 6, 7, 8

[5] Lior Wolf Benjamin Klein and Yehuda Afek. A dynamic convolutional layer for short rangeweather prediction. In *Computer Vision Foundation (CVPR2015)*. IEEE Xplore, 2015. 7

[6] Otakar Jašek. Detecting objects for autonomous system verification. Master's thesis, Czech Technical University, Faculty of Electrical Engineering, Department of Cybernetics, 2015. 7, 8, 9

[7] Jonathan Bhaskar. Image classification using convolutional neural networks. http://jonathanb.me/ImageClassificationUsingConvolutionalNeuralNetworksPaper.pdf. Accessed: August 2016. 7

[8] Philipp Käshammer. Interpreting visual data for ground and aerial robots. http://deeplearning.net/tutorial/lenet.html, 2016. 9

[9] Ladybug3 - datasheet. https://www.ptgrey.com/support/downloads/10149. Accessed: August 2016. 11, 39

[10] Laser sick lms-151 - datasheet. http://www.robotsinsearch.com/sites/default/files/produts/literature/LMS1xx/Product%20Information.pdf. Accessed: August 2016. 11, 20

[11] The robot operating system (ros). http://www.ros.org/. Accessed: August 2016. 11, 12

[12] Definition of "semantic". http://www.merriam-webster.com/dictionary/semantic. Accessed: August 2016. 13

[13] Tradr database architecture. https://redmine.ciirc.cvut.cz/projects/tradr/wiki/Database_architecture. Accessed: August 2016. 13, 14, 15, 24

[14] Vangie Beal. Definition of "database". http://www.webopedia.com/TERM/D/database.html. Accessed: August 2016. 13

[15] Definition of "database management system". https://en.wikipedia.org/wiki/Database. Accessed: August 2016. 14

[16] Mongodb. https://en.wikipedia.org/wiki/MongoDB. Accessed: August 2016. 14

[17] Stardog. https://en.wikipedia.org/wiki/Stardog. Accessed: August 2016. 15

[18] Resource description framework. https://en.wikipedia.org/wiki/Resource_Description_Framework. Accessed: August 2016. 15

[19] Pinhole camera technologie graphic. http://slideplayer.com/slide/5162869/. Accessed: August 2016. 20

[20] Opencv function project-points. http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. Accessed: August 2016. 21, 23

[21] 3d-plain equations. https://en.wikipedia.org/wiki/Plane_(geometry). Accessed: August 2016. 30

[22] Johann Borenstein, Lauro Ojeda, and Surat Kwanmuang. Heuristic reduction of gyro drift in imu-based personnel tracking systems. volume 7306, pages 73061H–73061H–11, 2009. 36