TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Persistent object mapping of indoor environments with the Toyota Human Support Robot

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao.Univ.-Prof. Dr. techn. M.Vincze
Dr. techn. T. Patten

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Christoph Berger
Wien, im November 2019

Vision for Robotics Group
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

Abstract

Mobile robots are becoming increasingly integrated into human environments. Especially in households and for people in need the use of mobile robots helps to improve everyday life. In order to interact with humans and to accomplish their various tasks the robots have to be able to locate obstacles and entities that are present in their working areas. Maps help to perform tasks like navigation and locating objects. Special techniques are required to classify objects and store additional information in the maps.

In this thesis the process of classifying, as well as locating objects and creating a persistent object map is presented. The persistent object map permanently stores the positions, sizes and shapes of detected objects and allows incremental information to be added. One of the fastest available detection algorithms is used and the gained information is transformed to create a three dimensional representation of the detected objects. The representations are permanently stored in a database and the stored information can be accessed even after a restart of the robot. The stored information is updated by new detections and multiple instances representing the same objects are replaced.

The process is implemented and tested on a human support robot from Toyota. Objects can also be classified and stored while the robot as well as some objects move and the locations of some representations are verified after a restart of the robot. Performance evaluations show the efficiency of creating the persistent object map in static and semi-dynamic environments.

Kurzzusammenfassung

Mobile Roboter werden zunehmend in meschnlichem Umfeld eingesetzt. Besonders in Haushalten und für Menschen mit besonderen Bedürfnissen können diese Roboter den Alltag verbessern. Um mit Menschen interagieren und ihre vielfältigen Aufgaben erfüllen zu können, müssen diese Roboter in der Lage sein, Hindernisse und Objekte in deren Arbeitsumfeld zu erkennen. Pläne helfen den Robotern bei Aufgaben wie Navigation und Lokalisierung von Objekten. Spezielle Methoden sind notwendig um die Objekte zu klassifizieren und zusätzliche Informationen zu speichern.

In dieser Arbeit wird ein Prozess zur Klassifizierung, sowie Lokalisierung von Objekten und zur Erstellung eines Nachhaltigen Objektplans (persistent object map) präsentiert. Der Nachhaltige Objektplan speichert dauerhaft die Positionen, Grössen und Formen von erkannten Objekten und ermöglicht die Ergänzung zusätzlicher Informationen. Einer der schnellsten Objekterkennungsprogramme wird verwendet und die erhaltenen Informationen werden transformiert um eine dreidimensionale Darstellungen von den identifizierten Objekten zu erzeugen. Diese Darstellungen werden dauerhaft in Datenbanken gespeichert und auf die gespeicherten Informationen kann auch nach einem Neustart des Roboters zugegriffen werden. Die gespeicherten Informationen werden mit neu erfassten Repräsentationen ergänzt und jene Repräsentationen, die dasselbe Objekt darstellen, werden zu einer zusammengefügt.

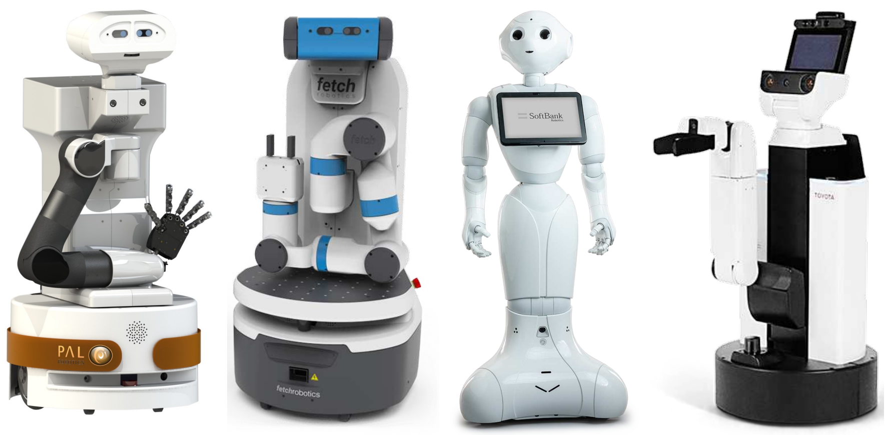
Der in dieser Arbeit präsentierte Prozess ist auf dem human support robot von Toyota implementiert. Objekte werden erkannt und gespeichert während sich der Roboter, sowie einige Objekte bewegen. Zusätzlich werden die Positionen einiger Repräsentationen nach einem Neustart des Roboters verifiziert. Eine Leistungsbewertung zeigt die Effizienz mit der der Nachhaltige Objektplan in statischen und semi-dynamischen Umgebungen erzeugt wird.

Contents

1 Introduction

Robots have evolved beyond the limits of industrial applications and are increasingly developed for usage among humans [1]. Apart from the well known robotic vacuum cleaners [2], mobile robots have been developed to perform different household tasks. Examples for these tasks are cleaning dishes [3], retrieving objects[4], supporting elderly people [4] or providing health care [5]. Four robots that tackle these tasks are displayed in Figure 1.1. A significant amount of companies, like Nvidia [6] or Temi [7], are currently developing mobile robots for indoor environments. These robots are particularly relevant in times of an increasingly ageing population that needs help for conducting household tasks [8].

For robots the collection of information of their physical environment is obligatory to accomplish their various tasks. The collected data is used to create an abstract representation of the real world, called a map. This map supports the robot to navigate in its environment and to access information about existing obstacles and objects [13]. Furthermore the map helps the robot to interact with the surrounding environment, manipulate present entities,



(a) Tiago, source: [9] (b) Fetch, source: [10] (c) Pepper, source: [11] (d) HSR, source: [12]

Figure 1.1: Household Robots

accomplish user queries or to reduce the response time of the queries [14]. One type is the object map, in which representations of detected objects are stored [15].

1.1 Persistent object maps

There are various tasks for a robot to accomplish when operating among humans. The success rate and response time for accomplishing these tasks can be improved by using object maps [15]. Elderly people are often in need of special objects and could relay on the service of robots to retrieve those objects. The robot has to locate the object and navigate towards it. This process requires a significant amount of computation time for the robot especially if neither the map, nor the current vision, include sufficient information to locate the requested object. The duration of locating and navigating towards requested objects can be enabled or improved with the help of persistent object maps, since persistent object maps provide the locations of all detected objects and obstacles to the robot[15]. Some requests of elderly people are difficult to accomplish with neither vision, nor prior knowledge of existing objects. For example if the elderly human wants the robot to retrieve a special cup next to a special bottle the relative position has to be calculated and the information of all present objects is useful. With the help of persistent object maps, which include the information of relevant objects, the success rate of accomplishing these tasks can be substantially increased

1.2 Encountered problems

Human beings and even animals are performing object recognition tasks day in and day out. They can, although not flawlessly, estimate the positions, sizes and the purpose of present objects by looking at them and locate them later if they want to use any of the seen objects [16]. To accomplish the same task robots need special hardware and algorithms to detect objects and estimate their locations, names, shapes and sizes. Special techniques are needed to collect the relevant information, which is obligatory to increase the success rate of correctly detecting the objects and making the correct assumptions [16]. The collected information needs to be saved and updated constantly by the robot to create and maintain an up to date persistent object map. Therefore the robot has to move through the room and constantly process images of the currently viewed scene.

A difficulty that is encountered while creating persistent objects maps is that the location of the robot and the location of the present objects are assumed to be static locations [17]. If the positions of either one changes the map has to be adapted in order to maintain the currently correct locations of the present objects, which have changed in relation to their prior locations.

False detection of objects or missing information represent additional challenges as they lead to false representations of present objects. This can happen because of missing depth information of shiny objects, pixels close to the edges of objects, or wrong assumptions leading to connections between pixels and objects that are not related [18]. The detection algorithm may also wrongfully name objects or create bounding boxes surrounding the objects, that are too big, or too small [19].

1.3 Object mapping process

In this thesis several problems are approached. One challenge is to gather the required information for estimating 3D bounding boxes out of 2D bounding boxes. Another challenge is the ability to access and update the information of relevant objects in the working area of the robot, even after restart or movement of the robot. Furthermore storing and modifying information about the relevant objects is approached. By the creation of a persistent object map these challenges can be approached.

Object mapping starts with the visual input from a camera, in particular from a camera that can detect color and distance to locate objects correctly within the real world. One of these cameras is the RGB-D camera, that uses the RGB camera and a depth camera which detect images consisting of the same amount of pixels. By aligning the cameras correctly the color and depth information of pixels can be obtained simultaneously.

Figure 1.2 displays the typical RGB images obtained by the RGB-D camera. A detection algorithm detects entities within the RGB image. These detection algorithms have improved over the last decades and there are some state of the art detection algorithms detecting objects from the input frame of a camera in real time, if the hardware specifications are sufficient. One of these algorithms, *You only look once (YOLO)* is used for creating the bounding boxes displayed in Figure 1.2. The bounding boxes in the 2D images are transformed to create 3D bounding boxes by processing the depth information from the depth camera. The sizes, shapes and locations of these 3D bounding boxes are estimated based on names, sizes and locations of the bounding boxes detected by *YOLO*.

After successfully detecting the present objects and creating 3D bounding boxes, the information is stored in a database in the form of a message. The stored messages are processed and filtered, for example to remove duplicate

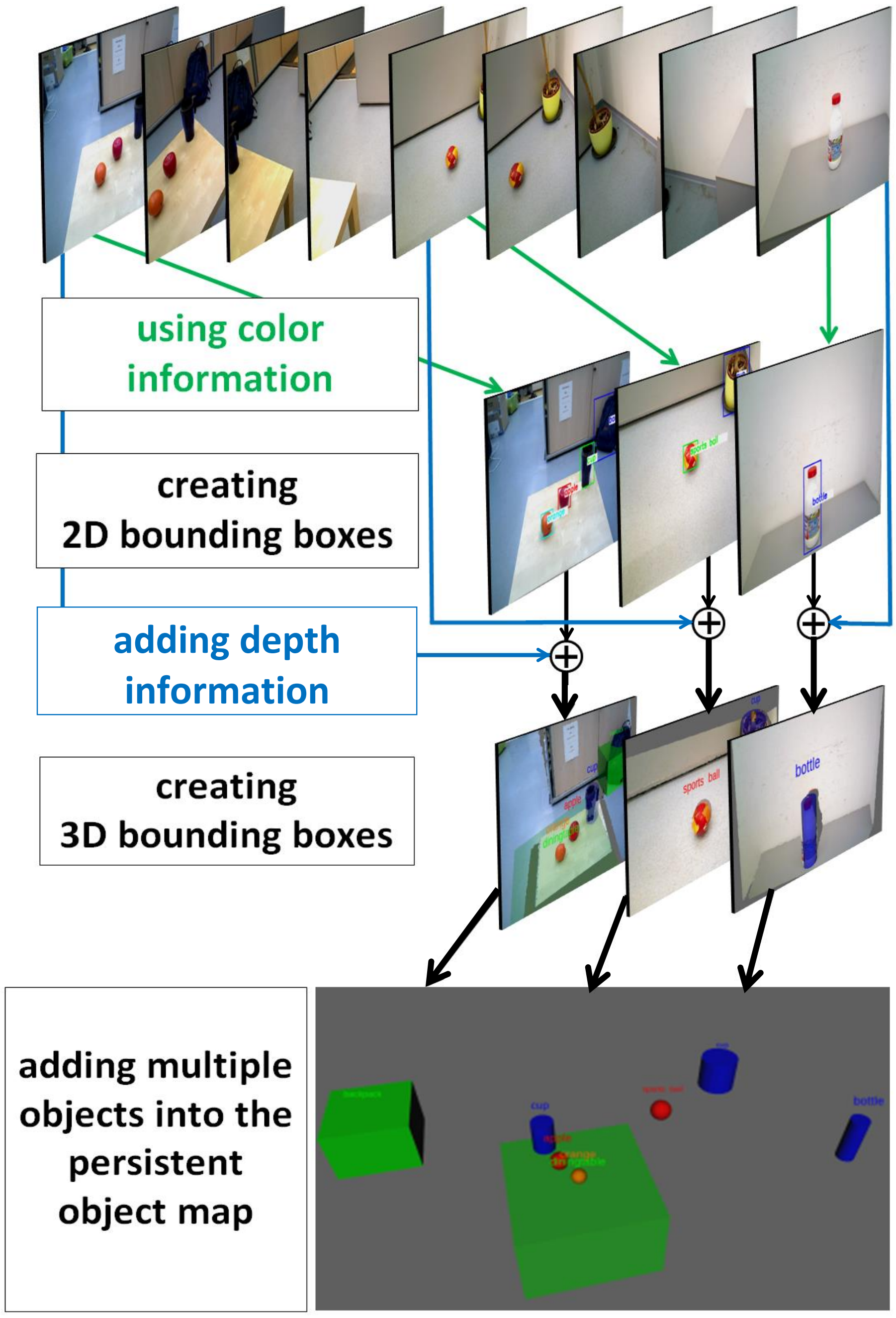

using color information
creating 2D bounding boxes
adding depth information
creating 3D bounding boxes
adding multiple objects into the persistent object map
sports ball
bottle
apple
cup

Figure 1.2: Creation of persistent object maps

detections of the same object. One result of the fusion, processing and filtering of stored messages in a database is displayed in Figure 1.2. The 3D bounding boxes with their shapes, sizes and locations are the representations of the detected objects within the working area of the robot. They are permanently stored in the database even after the shutdown of the robot, or the connected hardware. The processing and filtering of the 3D bounding boxes needs to be restarted after a reboot of the robot, or the connected hardware.

Creating persistent object maps with the provided hardware is designed to function properly for static environments, or slow movements of the robot or the detected objects. The programmed process is able to detect and locate present entities and visualize 3D bounding boxes, with basic shapes, surrounding the detected objects. The detected objects are permanently stored, new detections can be added and if only slow movements occur the process is able to replace multiple detections with the latest detection.

1.4 Thesis overview

The next Chapters will explain the creation process of the persistent object map thoroughly. Chapter 2 explains the already existing similar scientific approaches. In Chapter 3 a more detailed system overview is provided. The existing software which has been used is demonstrated in Chapter 4 and the programmed modules implemented in the creation process of the persistent object map are specified in Chapter 5. Chapter 6 describes the performed experiments with the complete implemented process. Chapter 7 provides an overview of the possible solutions for the encountered problems, as well as an outlook for future work.

2 Related Work

This Chapter outlines the differences of existing work related to the persistent object mapping algorithm in this thesis. In the first Section alternative methods for the object detection algorithm YOLO are elaborated. The second Section explains the mapping components employed in this thesis. Different forms of robotic maps and mapping processes are elaborated. The third Section describes related work that creates object maps and can store detected objects.

2.1 Object detection

The evolution of object detection methods led from detection algorithms that manually construct features to automatically feature constructing systems, the *Convolutional-Neural-Networks (CNNs)*. The evolution of CNNs, the most relevant CNNs and the object detection methods before CNNs are elaborated in this Section.

2.1.1 Manual feature construction

Object detection relies on the extraction of features, which are abstract representations of the whole input images or areas within the input image. The features are among others determined as global, when using the whole image for feature construction and local when features are constructed using areas of the image. Manually constructed features to describe the objects consumes a significant amount of time for the programmer. The accuracy for detecting objects correctly, same as the speed, is low in comparison to CNNs, especially for data sets containing a high variety of different objects [21] [22].

Local features

Local feature detection algorithms like *Scale invariant feature transform (SIFT)* [23] [24] [25], *Histogram of oriented gradients (HOG)* [26], or *Speeded Up Robust Features (SURF)* [25] extract local regions, with predefined sizes, around points of interest in the input image and transform every region into abstract representations.

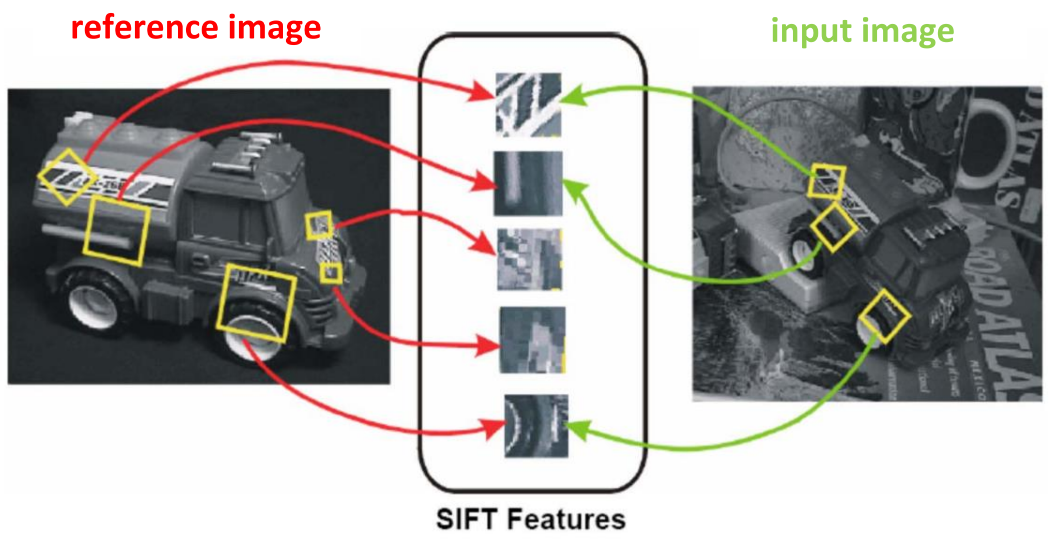


Figure 2.1: Local features with SIFT, source: [20]

Points of interest can be chosen in the gradient image, which consists of values that contain the directional change of the intensity or color of pixels compared to its neighbour pixels. Small values in the gradient image are removed by thresholds and the remaining values can form lines and when two lines are crossed corners are formed. These corners can be one form that determines points of interest. Further or other points can be chosen with different methods. Regions with the chosen points of interest in the middle are transformed into abstract representations. These abstract representations are local features and are local because only regions are determined to construct the features.

The local features are then compared to local features of reference images. The reference image can contain buildings, faces, or other forms that can be compared. For object detection the reference images contain objects. If a certain level of correlation between the input images and the reference image is determined the reference object is detected in the input image and bounding boxes with orientations can be constructed surrounding the detected objects in the input image.

One object detection algorithm with local feature construction worth mentioning is the Scale Invariant Feature Transform (SIFT)[24] [27], as the *SIFT* algorithm was widely used for local feature construction [28].

SIFT

With the development in 1999 the *Scale Invariant Feature Transform (SIFT)* algorithm used a form of local features that are invariant to image translation, scaling, rotation, and are only slightly affected by illumination changes [24] [27]. Reference features are constructed for a data set of objects and stored in a reference data set. The reference features can be compared to detected features. An example of correlating SIFT features in the reference image and in the input image is displayed in Figure 2.1. Even in cluttered images and only a few available features the algorithm is able to estimate positions, sizes and orientations of reference objects within an image [23] [27]. However compared to *YOLO*, SIFT and other feature extraction algorithms like SURF [25] or HOG [26] have many disadvantages. The manual construction of features is a time consuming and difficult process that can be done automatically by *CNNs*. CNNs also outperform the *SIFT* and other manual feature constructing algorithms in terms of the object detection speed [29] [19] [30] [31].

Global features

Global feature construction methods describe global features that capture the area of whole objects and not only predefined small regions, as the local features. Most commonly the global features are used in 3D object detection, geometric categorization and shape retrieval. Examples for manual global feature construction methods are the *Viewpoint Feature Histogram (VFH)* [32] , the *Shape Distribution on Voxel Surfaces (SDVS)* [33], the *Clustered Viewpoint Feature Histogram (CVFH)* [34] and the *Ensemble of Shape Functions (ESF)* [35]. Global descriptors observe the whole geometry of the 3D representations of the objects. 3D data, containing the objects, is obtained by depth cameras. For the construction of global features usually a prior segmentation of the scene has to be conducted where areas of possible object locations are obtained. The whole cluster is then described by a global feature and these features are then either classified as part of the object or of the background, which is not part of the object. The obtained global features includes information of the shape and texture of the detected objects. Creating global features can be unsuccessful when the objects are occluded or cluttered but the computation time and the memory expense is efficient [36] .

The manual construction of global features is, same as the manual local features construction a time consuming process and with the evolution of CNNs the manual construction of features can be performed automatically [37] [30] [19].

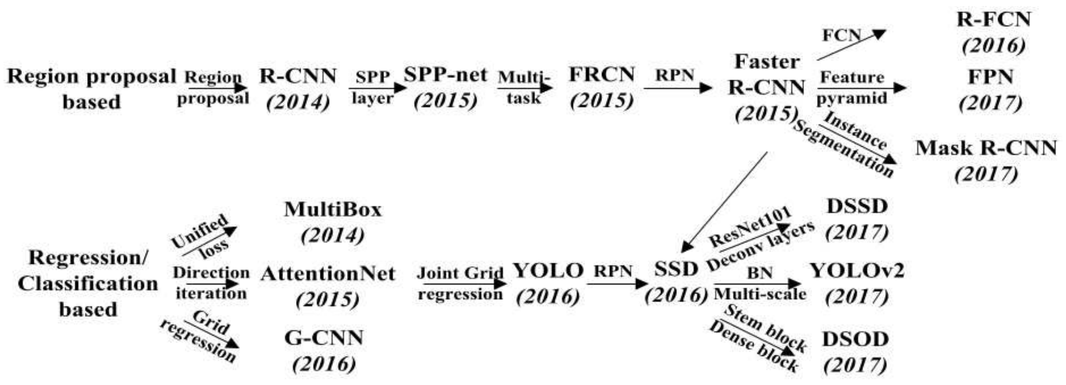


Figure 2.2: Evolution of CNNs, source: [22]

2.1.2 Convolutional neural networks

CNNs are a Deep learning method that can train features automatically and detect objects, speech, text and other recognisable information in a short period of time. They were inspired by the human way of processing images and the evolution of the CNNs is displayed in Figure 4.8. As displayed in Figure 2.2 during the evolution of the CNNs a number of algorithms using CNNs emerged and the most relevant are described here.

Alexnet

One of the first algorithms that used the implementation of a CNN is *AlexNet* [37]. *Alexnet* is used for classification of objects within RGB images. The algorithm resizes and crops the RGB images to 256x256 pixels and then randomly selects areas with fixed sizes to generate new smaller images. These images are used as input images for the first layer of the CNN. Each layer of the CNN filters the image with a predefined kernel and is, except for the last one, followed by a *Maxpool* layer. The *Maxpool* layer uses another filter, this time to determine the maximum value within the filter area. These layers create the features that are used for the classification of objects. The creation of features of reference objects takes days to complete and is called training of the CNN. The layers after the last *Maxpool* layer are the *Fully connected* layer and the *Softmax* layer. These layers determine if the features in the input image correlate to features of the reference objects. If there is a correlation the reference objects can be classified in the input image [37].

Regions with CNN features

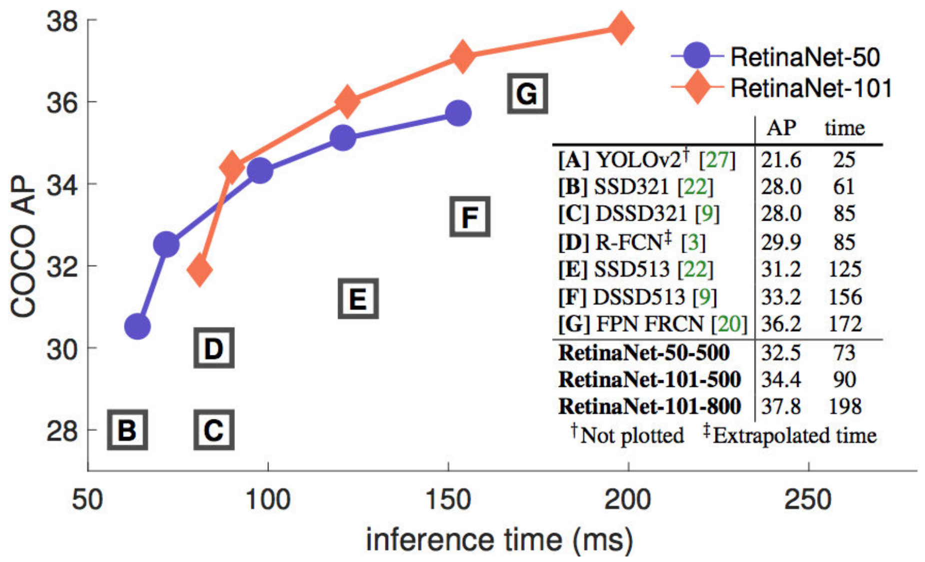
The *Regions with CNN features (R-CNN)* algorithm is one of the first object detection algorithms beside *Alexnet* that uses Deep learning for detecting objects and is therefore mentioned here. The RCNN uses a CNN that has five convolutional layers and two fully connected layers, it works with the same pricinple as *AlexNet*. The difference between *AlexNet* and *R-CNN* is that objects are not only classified but also localized and the input images are not randomly determined but are rather obtained by a *Selective search* method. The *Selective search* method groups pixels that have similar textures and then combines these groups until regions are constructed. From these regions images that define those regions are constructed and fed into the CNN. A typical R-CNN creates about 2000 regions and constructs about 4096 features for every input image. Then the class is determined and the bounding box is created by using a *Support Vector Machine* classifier and a *linear regressor* [30].

The *RCNN* algorithm is more than six times slower than the *YOLO* algorithm. *YOLO* already takes up to 8s to detect an image therefore *RCNN* could not be used as a real time object detection algorithm. There are several new versions of R-CNN as displayed in Figure 2.2. One improved version of the R-CNN algorithm *Faster R-CNN* [38] and also its predecessor *Fast R-CNN (FRCN)* [31] are still up to 6 times slower than the new version of *YOLO* and were therefore also not considered.

New versions of *RCNN*, *MaskRCCN* [39] and *R-FCN* [40] have the advantage of proposing segmentation masks for objects rather than only bounding boxes where also background information, which is region that is not part to the object, is included. The newer versions including *FPN* [41] can reach a very high accuracy but still take an significant duration more time to compute as displayed in Figure 2.3. Therefore the *RCNN* algorithms are not used and an other type of algorithms is used These types of algorithms do not use the region proposal stage, instead that are using their own approaches.

Single Shot MultiBox

The *Single Shot MultiBox (SSD)* [43] algorithm evolved after the first version of *YOLO* and had better overall performance in speed and detection confidence than *YOLO*. Different to the *YOLO* algorithm the *SSD* algorithm uses multiple *CNNs* and has a different approach for creating features. *SSD* uses different layers of the CNN to create feature maps. Feature maps generated at lower layers are used for detecting small objects and those generated at deeper layers detect bigger objects [43]. The disadvantage of this method is that features created in lower layers are not as high level as those created in deeper layers,



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

Figure 2.3: duration and accuracy of different detection algorithms, source:[42]

therefore detection of small objects with the SSD is worse that with other detection algorithms like *YOLO* and other *CNNs* [42].

The second version of *YOLO* outperforms the *SDD* algorithm in terms of speed and accuracy as displayed in Figure 2.3 and is therefore the chosen detection algorithm for object detection.

2.2 Robotic Mapping

There are two major frameworks for representing robotic maps, which are the metric and the topological framework. The metric framework saves the detected entities in a 2D map and the topological framework is a graph that can include the distances and other relations between entities and places. There are many forms of automatically creating maps and some maps are already created beforehand, like the metric map in Figure 2.4, especially for indoor environments.

2.2.1 Navigation maps

In Figure 2.4 a navigation map in different forms is displayed in addition to a semantic map and a topological map. A navigation map usually consists of two or three dimensional grids, where every cell of the grids can either be full or empty and it can be displayed in the metric framework and in the topological

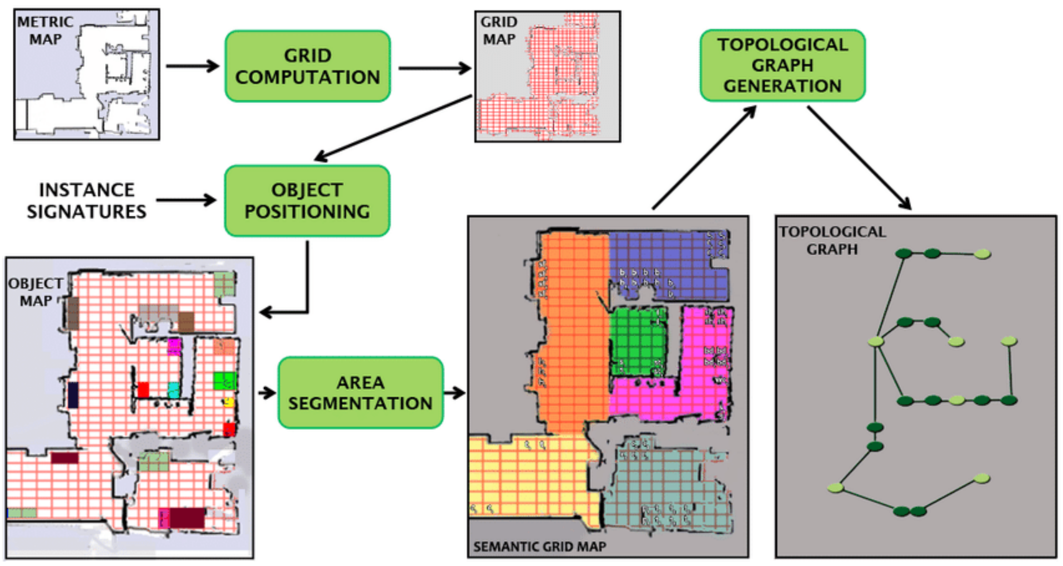


Figure 2.4: example for topological, semantic and metric map, source: [44]

framework. The grid map in Figure 2.4, with a two dimensional grid applied, is a navigation map and the object map can also be in form of a navigation map. Two forms of 3D navigation maps are the *voxel grid* with a fixed 3D grid size and the *cotomap* where the grid size is determined by the objects located in the grid cells [45].

The *voxel grid* map has a fixed gird size similar to the grid displayed in Figure 2.4, with the difference that *voxel grid* maps are usually three-dimensional. The single cells are also called *voxels*. Those that are containing whole objects, or parts of it, are marked as occupied and those that are not occupied are marked as free. The *voxel grid* has the disadvantage that the usage of the grids in not dynamic and every *voxel* has to be determined as either occupied or free [45].

The *octomap* [45] approached the problem of the static size of the voxel grids by including a dynamic form of size allocation. If an object is located in a *voxel*, which results in marking the *voxel* as occupied, the grid is divided into smaller *voxels* and the location of the object is determined again within the newly created smaller *voxels*. This process is repeated until a defined threshold is reached. More repetitions result in higher resolution of the objects but also increased memory usage and computation time. The navigation map can also be displayed as a topological graph, with for example occupied voxels marked as black and free voxels marked as white. The *voxel grid* map and the *octomap* are well suited for navigation, because the trajectory for robot movements can be planned by avoiding occupied *voxels* and using only the space where the *voxels* are not occupied [45].

The duration for creating a persistent object map would have increased by a significant amount of time if *voxel grids* were used, therefore these are not implemented. Instead a database is chosen to store the detected entities, because of the easy implementation into the *ROS* framework and because of the dynamic storage of *ROS* messages leading to easy additions and queries of stored entities. The *SQL* and *NoSQL* databases in addition to the used database, *mongoDB*, are explained in Section 4.4.

A navigation map defining the surrounding environment already exists and this map could be used in addition to the persistent object map, as the example in Figure 2.4 shows. The location and orientation of the robot in space is, due to internal location and orientation processes, not or only under certain circumstances consistent with the metric map and the objects would be placed at the wrong positions. Therefore the existing metric map is not used.

2.3 Persistent object mapping

Besides the navigation maps there are semantic maps that can be used for creating an object map. The semantic map can make use of *octomaps*, *voxel grid* maps or other grid maps. The occupied *voxels* can be grouped and a semantic map can be created that includes 3D models of objects consisting of occupied *voxels*. These 3D models can be labeled and the created semantic map results in a map consisting of labeled 3D models. An example of a semantic map is displayed in Figure 2.4, but the labels are not displayed. There are several forms of semantic maps, like *semantic fusion* [46], or *fusion++* [47] but same as the navigation maps the existing algorithms for creating these maps take too much computational effort to be considered for implementation.

Other approaches introduce different forms of persistent object mapping, which combine the detection of objects, as well as the mapping and the permanent storage of these detections and maps.

The approach in KnowRob [48] uses the information of other detecting algorithms and forms a persistent object map. RoboSherlock [49] uses the information from KnowRob and improves the persistent object map. These systems create a persistent object map but rely on an existing object detection algorithm.

The approach of [50] constructs persistent object maps directly from the pcl with self designed models. Constructing the models takes a significant amount of effort and the detection duration is not able to achieve real time detection.

3 System overview

In this Chapter a overview of the used hardware and the programmed algorithms is described. Figure 3.1 shows the creation process of object maps in the form of nodes. Nodes are programmed sub processes that receive incoming data from other nodes, or sensors, process the incoming data and send out new data. Which nodes are running on the robot and which are running on the computer is marked in Figure 3.1. The computer and the robot communicate through a wireless network.

The first Section of this Chapter describes the hardware specifications. The second Section presents an overview of the object mapping process and the implementations of the nodes displayed in Figure 3.1.

3.1 Hardware

3.1.1 Robot

Toyota provides four main classes of robots as shown in Figure 3.2, which are the Rehabilitation, the Social, the Human Support and the Innovation Robots [51]. These robots are in development stage and only a limited number of each has been produced. The Rehabilitation Robot should assist people to recover after diseases or other circumstances that caused leg paralysis. The Social Robot should improve the life of elderly people by engaging in conversation. The Innovation Robot is the third generation of humanoid robots provided by Toyota [51]. The robot used for the object mapping process is the *Human Support Robot (HSR)*. The *HSR* should help to promote the independence of people in need, for example to retrieve objects or provide household support [4]. The components of the *HSR* are described in Figure 3.3.

The *HSR* has an internal computer controlling the components displayed in Figure 3.3 and additional sensors and actuators that are not displayed. The operating system of the computer on the robot is Ubuntu 16.04 with *ROS-KINETIC*. The CPU of this computer on the robot is the i7-4700EQ with 2.4 GHz and 8 GB of RAM are provided.

As displayed in Figure 3.1 the OM-N and the camera-Node run on the robot. Due to the high calculation expanses of image processing for the object

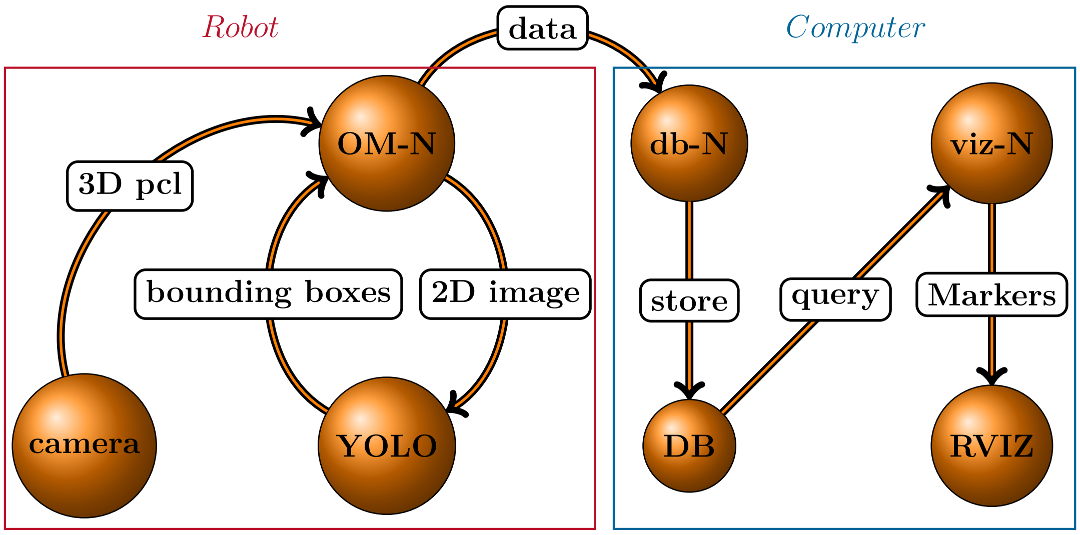


Figure 3.1: Nodes on the robot and nodes on the Computer

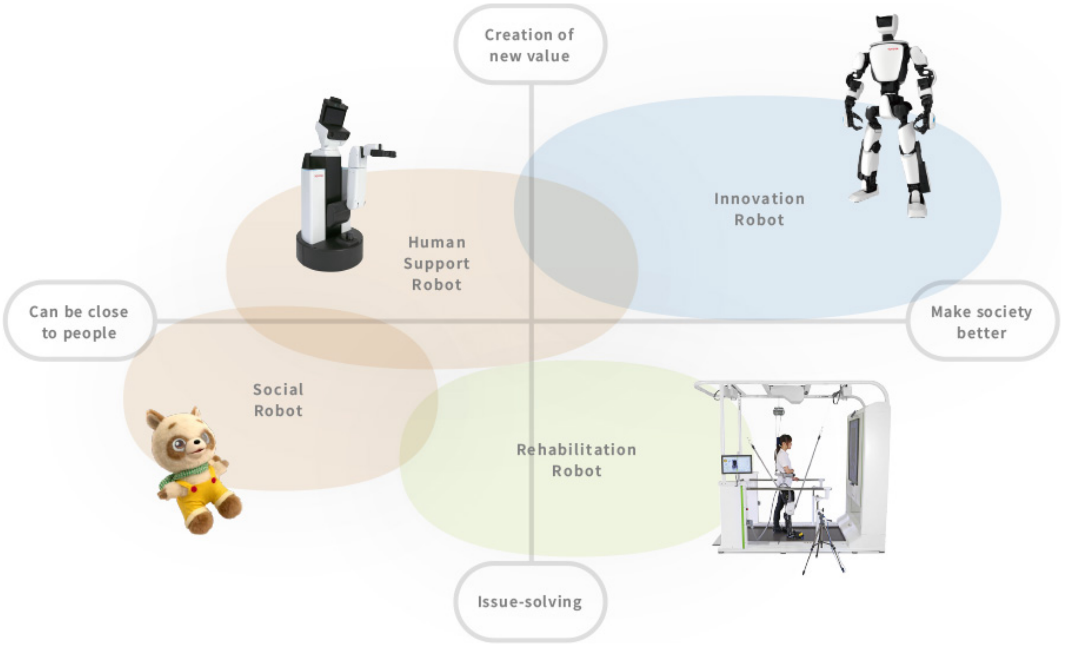


Figure 3.2: Human support robot, source:[12]

detection algorithm *YOLO* the computer needs a graphic processing unit to work fast. The Jetson TX1 module is used as GPU and a second Ubuntu system is installed, which uses the Jetson TX1 module. This system is the 14.04 Ubuntu system with CUDA 6.5 and *ROS-INDIGO* installed. On this system only the *YOLO* node from the robot frame in Figure 3.1 is running.

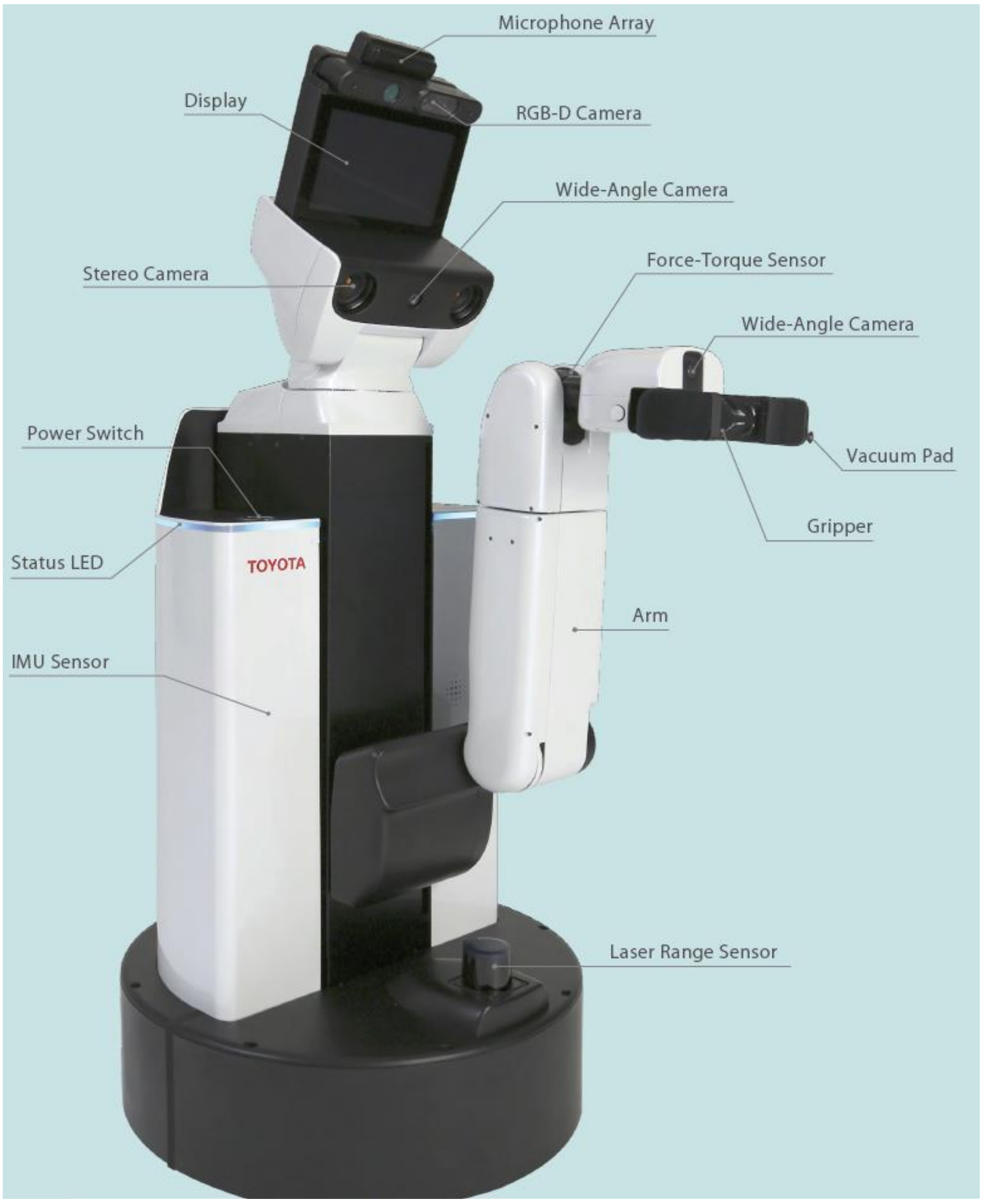


Figure 3.3: Human support robot, source:[12]

For detecting the depth and RGB images the Xtion PRO LIVE camera from ASUS is installed, this camera is marked as RGB-D camera in Figure 3.3. There are different RGB cameras available on the robot, for example the Wide-Angle camera, the Stereo camera and for the detection of depth images a Laser Range Sensor, but only the RGB-D camera provides a depth camera and a RGB camera that are aligned to each other. The robot can move with a maximum speed of 0.8 km/h, the Laser Range Sensor located at the bottom can measure distances towards obstacles while moving and a bumper sensor at the bottom can detect collisions.

11 actuators are installed to provide movements of parts of the robot. The neck of the robot can move in two *Degrees of Freedom (DoF)*, the arm can pick and place objects in 5 *DoF* with 1 *DoF* from the gripper. With the camera and the suction pads on the gripper the robot knows when it has grasped an object and can lift objects like photographs that otherwise cannot be lifted. The base has 3 *DoF* and the Torso 1 *DoF* which enables the robot to move through the room and the body to reach a height of 135 cm.

An additional computer is used apart from the robot to outsource some processes and store data.

3.1.2 Computer

As displayed in Figure 3.1 there are three nodes running on the robot and four nodes running on the Computer. Any computer can be used to run those nodes as long as its hardware specifications are sufficient. For the tests conducted in Chapter 6 a computer with a i5-2500 CPU, the GeForce GTX 1050 Ti graphics card and 8 GB of RAM has been used. The communication between the Robot and the computer is established via the wireless LAN router Linksys EA 4500.

3.2 Software Implementation

The goal is to perceive the positions, names and sizes of the objects within the working area of the robot and to store them into a database. In order to get the correct positions of the objects a camera with the ability to measure distances is mandatory, therefore a depth camera is used. The depth camera uses an infrared pattern projected in the relevant area in combination with an RGB-camera and creates a so called *Point-Cloud (pcl)*. The pcl consists of pixels and has, in addition to the RGB-values, information about the positions within 3D space for each pixel.

As shown in Figure 3.1 in the first step the Object-Mapping-Node (OM-N) receives the pcl data from the camera.The OM-N uses the 2D RGB image of the pcl and sends it to the *YOLO* node. *YOLO* uses the 2D image to detect objects within the image. The result of *YOLO* is represented by the bounding boxes data which is sent back to the OM-N. By combining the pcl and the bounding boxes data the positions, sizes and names of the present objects can be estimated. The data from the detected objects is transformed and stored into a database by the *data-base-Node (db-N)*. The database permanently stores the transformed data with the detected objects.

For the visualization of the stored objects a query of the database is imperative, which is done by the visualization-Node (viz-N). Received data, from the query of the database, is sorted and filtered by the viz-N and transformed into a marker. A marker is a 3D representation of an object, with size, shape and location.These markers can be visualized with a visualization program which displays them in addition to the current camera view. After restart of the robot, connected hardware or any movement of the robot the markers remain at the same location defined within the database.With the permanent storage of the locations of the detected objects the created object map can be classified as persistent object map.

This overview of the process is separated into two parts and explained thoroughly in Chapters 4 and 5. The first part in Chapter 4 explains the existing software that is used for the process and the second part in Chapter 5 explains the programmed nodes.

4 Supporting Framework

This chapter describes the existing software used for the creation of the object map. In the first Section of the Chapter the *ROS* framework is explained. The framework is used to program the existing nodes displayed in Figure 4.1. Figure 4.1 shows the same nodes as Figure 3.1 with the addition of the separation between the existing software and the programmed software. The *ROS* framework also provides the program RVIZ that is used to visualize 3D environments. Visual input is the second Section which elaborates the methods for creating *(3D pcl)* messages by the fusion of the RGB image and the depth image input from the camera. The creation and transformation of point clouds is also elaborated in this Section. The third Section explains the evolution of the object detection, deep learning algorithms and the used deep learning object detection algorithm *YOLO* . In the last Section the *mongo-database (mongoDB)* is explained and SQL as well as NO-SQL databases are elaborated.

4.1 ROS Framework

This Section describes the nodes of Figure 3.1 and how the transportation of messages is managed.

Robot Operation System, short *ROS*, is a framework established in 2007 [52]. *ROS* is, contrary to its name, not an operating system, but rather provides a communication layer operating above the operating system [53].

There are four concepts implemented in *ROS*: nodes, messages, actions and services. Nodes are programs written, most commonly, in the languages Python and C++. Communication between the nodes is established by the transmission of messages which can consist of standard types like integers, floating point, boolean but they can also include arrays and other messages. The way a message is transferred, from one node to another node, is through topics. One node can publish messages to multiple topics and subscribe to multiple topics. Also more than one node can publish messages to one topic or receive the message from one topic. The service and client concept is another way to transport information. The service node receives a call from a client node in form of a service file. The service file includes the request and the response data types. Once a service node receives a call with request data from

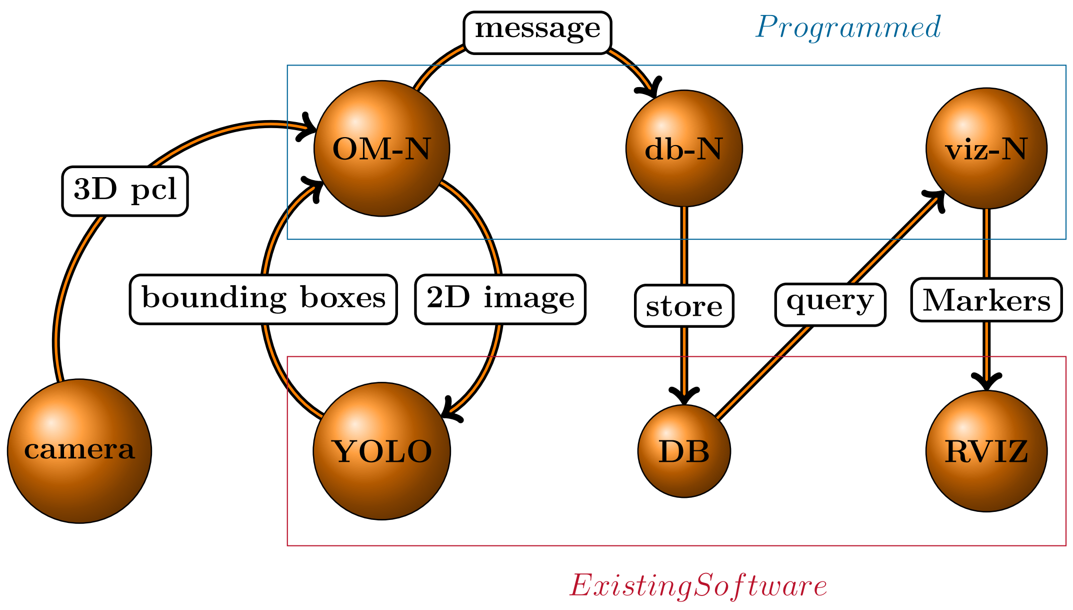


Figure 4.1: Programmed Nodes and Existing Software

a client node, it calculates the data required for the response once and then sends the response data back to the client node. Multiple clients can call one service node. The *ROS* action is another form of concept that is similar to the *ROS* service. The action concept also consists of the server and client construct with the difference that the request can be canceled and a feedback is added to the definition of the communicated message. The feedback provides the client with information about the progresses of a goal [52]. With these basic elements complex systems can be constructed and displayed in form of a map by *ROS*.

4.1.1 Nodes and topics

If for example all the nodes and connections of a mobile robot are displayed, the number of connections is too high to extract useable information by a human. However by enabling only part of the nodes and topics, the graph can be reduced to only the important entities. This reduction has been done for the nodes used in this thesis, by excluding the nodes and topics that are not relevant and then displaying the remaining, relevant nodes with *ROS*. Referring to Figure 3.1, which is a created Figure for the overview, the overview can also be generated automatically as a graph by *ROS*, as displayed in Figure 4.2. The graph shows the topics where the messages are sent to and subscribed from, which are displayed surrounded with a rectangular bounding box. The nodes in the graph are displayed surrounded by an oval box.

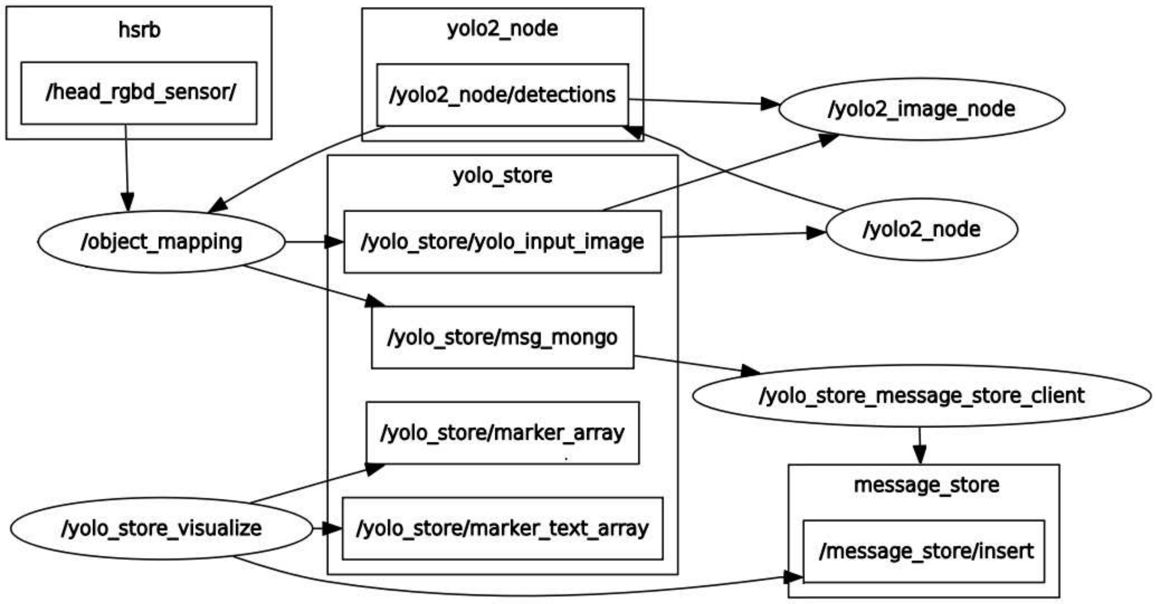


Figure 4.2: Graph generated by *ROS* with topics and nodes

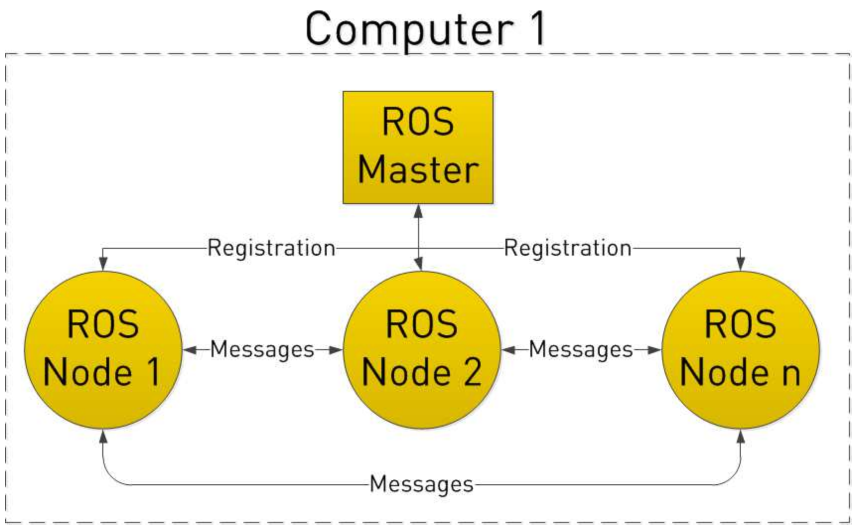


Figure 4.3: Nodes connected to master communicating with messages, source:[54]

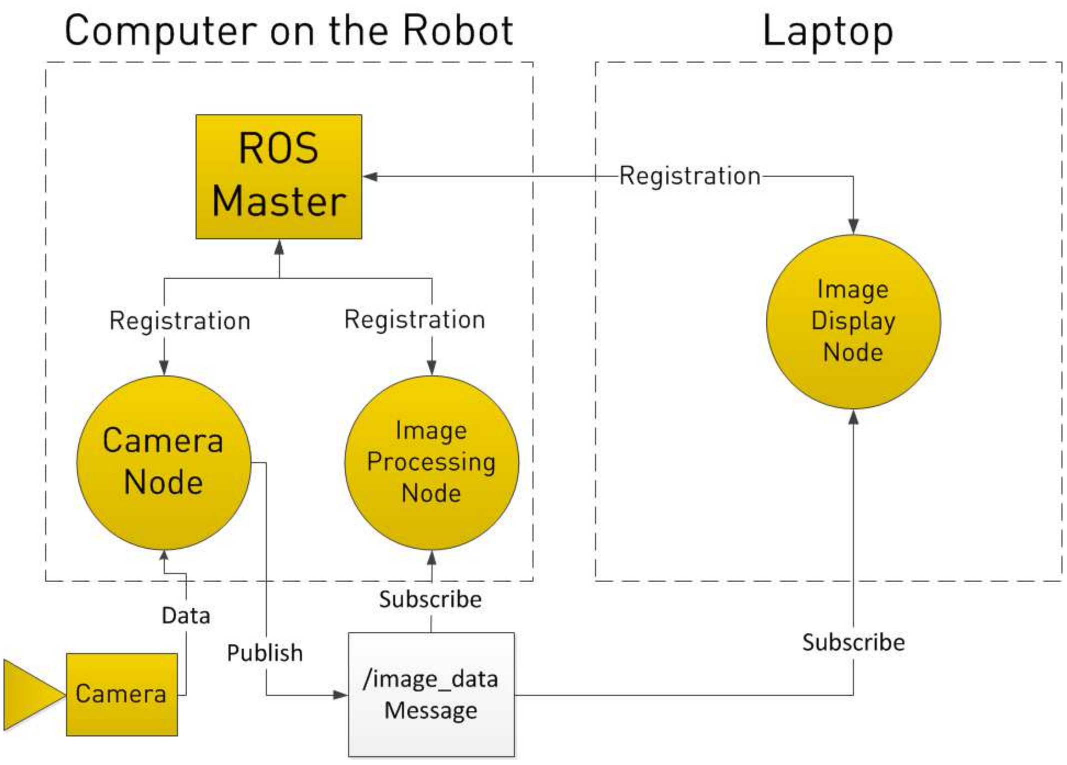
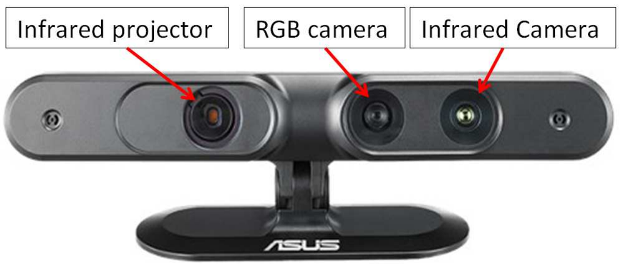


Figure 4.4: Communication of nodes with two operating systems, source:[54]

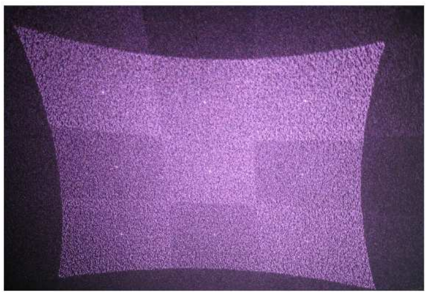
Unfortunately the database server, as well as the visualization program, cannot be included with their connections and the workflow is not displayed sufficiently enough, therefore Figure 3.1 was created for a better overview. Additionally in Figure 4.2 the topics, rather than the transported messages, are displayed which also reduces the overview. One more element though, that is neither included in Figure 4.2, nor in Figure 3.1 is the transform topic. The transform topic is used for transform function messages, which include relevant reference coordinate systems of the robot. With the usage of these transform function messages points and objects, detected within one coordinate system, can be transformed to another coordinate system which is explained in Section 4.2.3.

4.1.2 Communication of nodes

If all the packages are built and executable, the architecture of *ROS* is important to understand before the nodes displayed in Figure 4.2 can be started. All nodes started with *ROS* have to be registered and named by the master node as displayed in Figure 4.3. The master node tracks all available topics, as well



(a) Asus Xtion PRO LIVE camera, source: [55]



(b) Infrared pattern, source: [56]

Figure 4.5: RGB-D Camera

publishers, subscribers and services. With the master the nodes are able to locate one another at run time and can then communicate peer-to-peer. The peer-to-peer communication reduces the traffic flow across wireless links and therefore increases the speed of communication [53]. The master node and the nodes connected to the master node, can run on the same operating system, but that is not mandatory. The master node does not have to run on the same operating system as the other nodes, it can be exported from one operating system to another, the only necessary condition is, that the operating system must be connected to the same network. In Figure 4.4 an example for the communication of two operating systems, one on the robot and the other on the laptop, is displayed. The camera sends the collected data to the camera node, which creates a message out of the incoming data and publishes it. Figure 4.4 also describes the basic way how the data of a camera is transformed into a message and published. The created message and the visual input in general are explained in the next Section.

4.2 Visual Input

In the 1970s the first attempts to recover the three dimensional structures of objects and surrounding environments were made. The first methods, that are still partly used, gather information of the brightness of pixels within an image and estimate the boundaries of objects by the change of the intensity of the brightness. Positioning of objects within environments is made possible by the use of range sensors that can locate the position of pixels in three dimensional space.

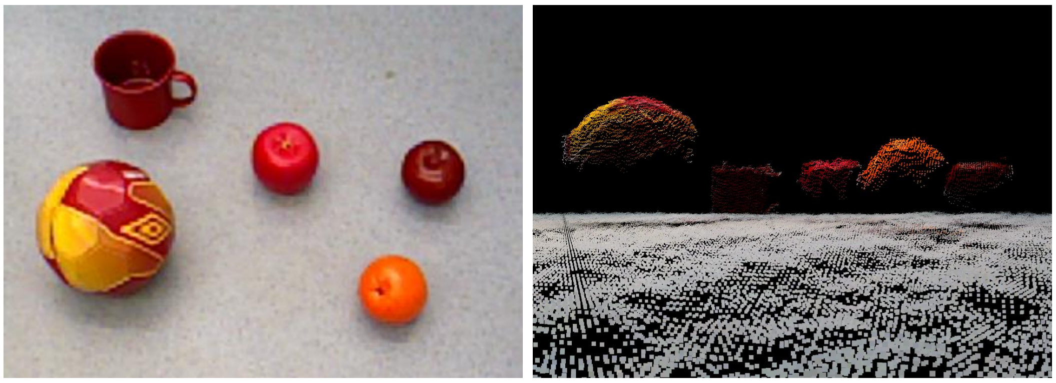

(a) Input RGB (b) Pointcloud

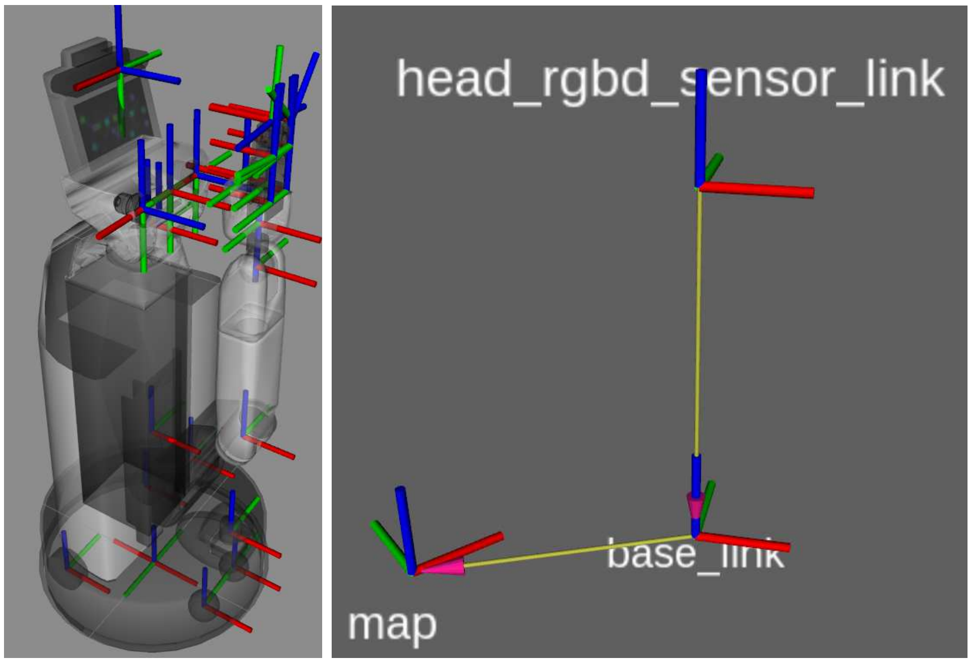
Figure 4.6: RGB and pointcloud

4.2.1 ASUS RGB-D camera

In order to detect the positions and gather the color information of objects the range information and color information has to be acquired by a RGB-D camera. One example for a RGB-D camera is the *Xtion PRO LIVE* built by the company ASUS, displayed in Figure 4.5a. The *ASUS Xtion PRO LIVE* is a RGB-D camera that uses two cameras that can detect images with a resolution of 640x480 and a frame rate of 30 frames per second(fps), or half the resolution and double the frame rate. One of the two cameras of the RGB-D camera from *ASUS* is a RGB camera that is similar to cameras on mobile phones or digital cameras. This RGB camera detects the color values of every pixel. The second camera is an infrared camera that uses infrared patterns, projected by the infrared projector, to measure the distance from the camera to the pixels. The infrared pattern of the infrared projector is displayed in Figure 4.5b. The RGB camera is aligned with the infrared camera to create both color and depth information of every pixel and the acquired data is stored in the form of a pcl.

4.2.2 Point cloud

The colored pcl is created from the RGB image captured by the RGB camera in combination with the image from the infrared camera which are both displayed in Figure 4.5a. Both cameras have to be aligned correctly to view the same pixels when assigning the color and depth values to the pixels. The *Point cloud library(PCL)* provides necessary algorithms required to process the input images, with the depth and RGB values, in order to create a *pcl message* The elements of the *pcl message* are points instead of pixels like in a RGB image. This *pcl message* includes the color values red green and blue (RGB) for



(a) All Robot frames (b) Frame transformation from camera to map

Figure 4.7: Frames of the Human Support Robot displayed in RVIZ

every point as well as the position in 3D space with the x,y and z coordinates. Furthermore the message includes the information of the frame that took the pcl and is used to transform the coordinates of the points from one frame into another.

The name of the pcl originates in the points stored in the pcl that look like a cloud created out of points if visualized [52] [57]. An example of a pcl taken with the *ASUS* depth camera, as displayed in Figure 4.5a and visualized with the program *RVIZ*, is displayed in Figure 4.6b. Figure 4.6a shows the RGB image viewed by the RGB-camera used for creating this pcl. The camera looks at the objects from above and partly from the side, therefore these sides of the objects can also be captured by the infrared camera including the surrounding floor. The bottom side of the objects is not viewed by the camera and therefore cannot be visualized. The output pcl with the input RGB image is displayed in Figure 4.6

4.2.3 Transformation

The pcl message has a reference coordinate system, also called a frame that can be used for transforming the (xyz) values from one coordinate system into another. The relations of these frames are provided by the transform message, published by the camera node. The relations between the camera frame and the map frame are displayed in Figure 4.7b. All available frames of the HSR are displayed in Figure 4.7a.

The transformation of the coordinates is done with the rotatory and the translatory information from the transform message entered into the transformation matrix. This matrix is then multiplied with the 3D coordinates of a point of the pcl to receive the coordinates of this point within the desired frame [52].

4.3 Object detection

Object detection is a fundamental computer vision problem helping machines and robots to understand their environment. The evolution towards deep learning algorithms, with large data sets for the training of the programs and increasing computation performance, enabled the processing of images in real time with a high success rate of correctly detecting objects [22].

4.3.1 Machine learning

Over the last decades the main method for detecting objects is *Machine Learning* which uses the data retrieved by cameras and other input methods and transforms it into useful information. In this transformation the data is processed to detect features represented by vectors or other forms which can be used by a secondary system for further computing. The secondary system can then detect certain features which, in combination with a pattern recognition, enable the retrieval of information about the present objects [29].

For humans the capturing of three dimensional entities around them is a everyday procedure. It is easy for humans to make the correct assumptions about the positions, sizes, shapes and the designated names for the objects that are present in the environment. Even if these object have never been seen before the classification as a certain type is possible for humans. For example if a mug, an apple and a plate are lying, or standing, on a table we can presume the purpose of the table and the purpose of the objects. The table will most likely be a diningtable, the apple will be eatable and the plate as well as the mug will be used to be filled by other forms of objects or liquids. Although there are some situations, for example optical illusions, where humans fail to make the correct assumptions about length, size, or colour the assumptions of humans about objects are correct and can be made with ease. The example of detecting objects on a table is a everyday procedure for humans and they

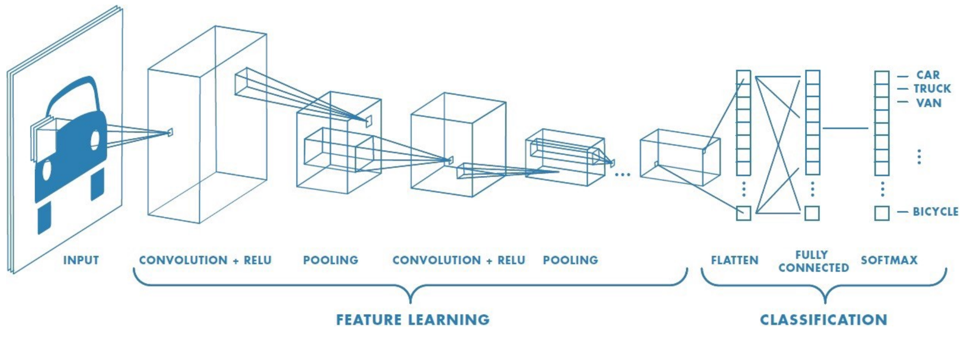


Figure 4.8: Convolutional neural networks, source: [59]

automatically train these scenes while growing up. Even for a young child the counting of animals or exotic fruits within an image, and evaluating the positions of the counted animals or fruits, can be solved, assuming the child can count.Given the same tasks object detection algorithms have problems with making even part of the same assumptions compared to humans [58].

A computer has to transform the incoming image to extract features out of the image and compare it to other reference images to understand the scene and detect present objects. Constructing robust features that allow a high success rate for detecting objects in a short period of time is a complex process that can be done automatically by so called deep learning networks [29].

4.3.2 Deep learning

Deep learning architectures, such as (CNNs) use multiple layers to extract high level features that are used for object detection. Low level layers may consist of basic elements like edges or corners and high level features may contain abstract representation of objects [29].

One evolutionary step towards Deep Learning is *Representation Learning*. *Representation Learning* can transform the input data from the cameras, or other input methods directly into the information needed for detecting objects. With multiple non linear transformation of the input image the deep learning algorithm can retrieve information about present objects within the input image. The main computing time for detecting objects with deep learning algorithms is taken for the training of the algorithm. The success of object detection is measured by a score that determines the correctly detected objects and during training, a data-set of images is processed that results in certain parameters of obtained high level features. The error of the achieved scores compared to the detected scores must be reduced by adjusting the parameters. Training times are measured in hours and can last for several days depending on the amount of trained objects during that period [29]. After the training the algorithm can work very fast and detect objects of incoming images in real time depending on the hardware specifications of the computer and the rate of the incoming images per second [19].

4.3.3 Convolutional Neural Network

CNNs are deep learning architectures and are structured as displayed in Figure 2.2. The input RGB image is filtered with a kernel that has a defined size and a new layer is created. This is done to reduce the computing power required to process every pixels of the image. This newly created layer can be and is usually filtered again, this time with a pooling algorithm. This pooling algorithm uses the filter to determine the maximum or the average value within the processed filter. This process is repeated for a number of times, depending on the algorithm and for every repetition a new layer is created. More layers extract richer and higher level features that typically result in more successful detection but require a higher computational effort and duration. There is a certain limit for the number of layers after that newly added layer will not improve the detection rate any more. These layers are used for training the deep learning network, in other words to help the CNN algorithm to understand the high level features that are obtained and to associate them to objects classes [59].

The last layer of the feature learning layers is flattened into a vector. The size of the vector depends on the size of the input image, the number of layers and size of the filters. Usually the flattened layer is then connected to one *Fully Connected (FC)* layer and one *Softmax* layer. These layers are responsible for the detection and classification of objects by using the extracted features from the convolutional layers [59].

4.3.4 You only look once

The CNN that is used for detecting objects in this thesis is *YOLO* and three versions of *YOLO* currently exist [19] [60] [61] [62]. All versions of *YOLO* use the *Darknet* [63] framework and are trained on the data set of *ImageNet-1000* [64] [65]. The first version of *YOLO*, *YOLO v1*, resizes the input image, uses one CNN and a non-max suppression to detect objects present in the input image. This process is displayed in Figure 4.9 [19]. *YOLO v1* uses 24 convolutional layers followed by two fully connected layers for the CNN. [19] The main difference between the *RCNNs* and *YOLO* is the number of stages used for extracting the features. *RCNN* uses two stages, the region proposal stage and the detection stage, whereas *YOLO* only uses the detection stage

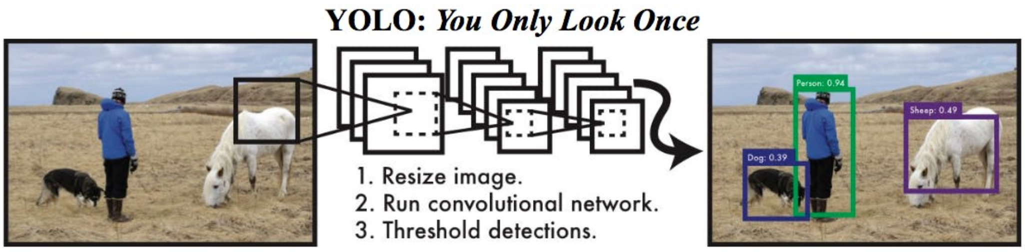


Figure 4.9: Workflow of YOLO, source: [19]

with one CNN. *YOLO* divides the input image into grids and predicts class properties for every grid. Every grid creates a number of bounding boxes and the results are class specific scores for every bounding box. *YOLO* can reason globally about detected objects and classes in the image, which the *RCNN* algorithm does not, because the *RCNN* only uses areas of the image and not the whole image for detecting objects. A major advantage of *YOLO* is the high speed of the algorithm that achieves real time object detection and can, besides for indoor households, be used in the field of autonomous driving and other time critical environments [66].

With the second version of *YOLO*, *YOLO v2*, the speed of the algorithm improved compared to *YOLO v1* and is presently one of the fastest algorithms in the object detection area [62] [60]. The major improvements of the second version of *YOLO*, apart from the speed, were higher success in detecting small objects and better overall detection rate of objects. The third version improves the *YOLO* algorithm further, but due to the lack of achieving the minimal hardware requirements *YOLO v3* could not be used with the *HSR*, therefore the *YOLO v2* is used and referred to as *YOLO* in this thesis.

The resulting output created by all the versions of *YOLO* is a bounding box.

After the objects, present in the 2D image, are detected by the CNN and the non max suppression has been applied a bounding box is constructed. This bounding box is defined by its name, the confidence percentage, the coordinates of the center point, the length and the width. The coordinates, as well as length and width, are measured in pixels and can be displayed inside the processed 2D image. One example for a detection of *YOLO* with the names and bounding boxes surrounding the detected objects is displayed in Figure 4.10. All the bounding boxes of the detected objects are fused into one message and published by *YOLO* as a bounding boxes message.

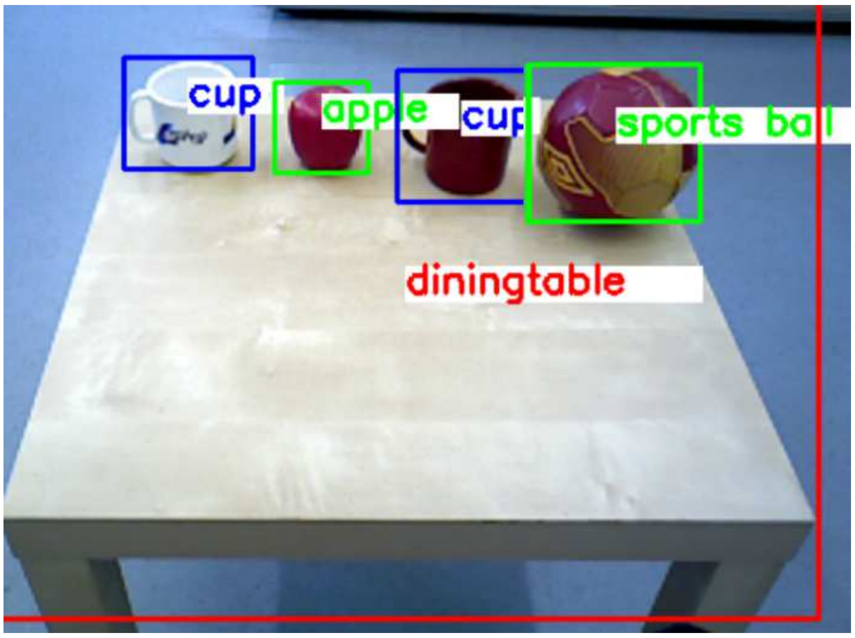


Figure 4.10: Detection sample of YOLO

4.4 Database

A database collects data and information in form of bits and bytes. This data is updated and manipulated through the Data definition language and the data manipulation language which are combined into one language. One example of this combined language is the *Structured Query Language (SQL)*, which is the ANSI standard for manipulating and accessing information stored in a relational databases. Data is stored in form of tables and is usually structured in relational databases and named SQL-databases. SQL databases are tables with n rows, are vertically scalable, which means that better hardware results in better speed and they have a predefined schema. The non relational database with the language *Not only SQL (NoSQL)* stores data in form of documents and the stored data is usually non-, or semi-structured and called NoSQL database. NoSQL databases can have a dynamic schema and store messages with key value pairs, that means that data can be easily added and removed. Especially for adding different information to the already stored data and constantly being able to update and change stored data the NoSQL database is the preferred database over the SQL database. Therefore the NoSQL database mongoDB has been chosen, also due to the fact that an implementation of mongoDB into the *ROS* environment already exists. An example of how data is stored in

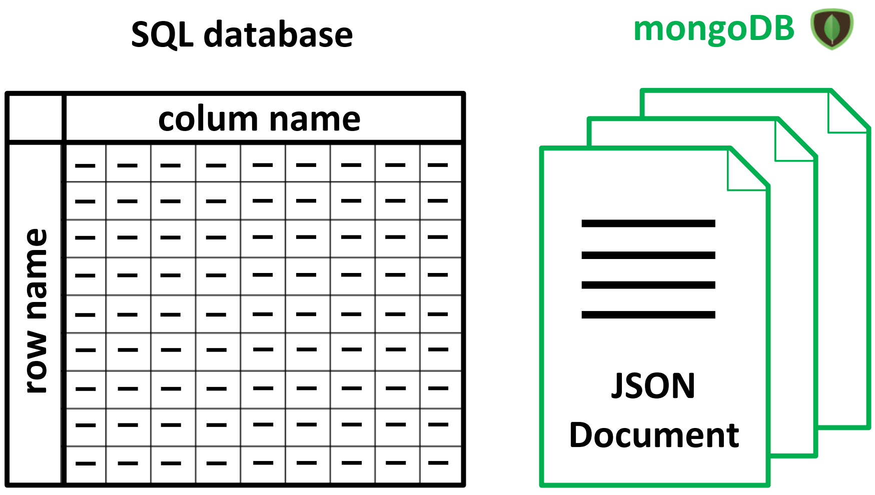


Figure 4.11: SQL and mongoDB data management, source:[67]

form of documents in the mongoDB and in form of tables in a SQL database is displayed in Figure 4.11. The lines are symbolic representations for stored data.

4.4.1 MongoDB

The mongoDB uses the Binary JSON (BSON) structure for storing information inside the database. The BSON is a simple representation of data and includes only six datatypes, boolean, integer, float, date, string and binary types. There is an implementation of the *mongoDB* called the mongodb_store provided by *ROS* [52]. This implementation can store predefined or newly created *ROS* messages via a command into the database.

The node responsible for storing these messages is the database-Node (db-N). The messages stored into the *mongoDB* have to include an id for unique identification. All incoming messages from the db-N are collected by the db-N and stored into the currently active mongoDB. The *mongoDB* is created in a folder and has to be started with a command from *ROS* to be activated. Different mongoDB databases can be created in different folders for example to only store certain types of objects. The stored data can be accessed via a query that can request all stored messages in the mongoDB and the objects included in the messages. The query to access the information of the stored objects is conducted by the viz-N.

5 Persistent object mapping

This Chapter describes the programmed code, which, as seen in Figure 3.1, are the OM-N the db-N and the viz-N, they are explained in Sections 5.1 through 5.3. These nodes are all programmed as part of the *yolo_store ROS* package. Figure 3.1 displays the workflow of the object mapping process, including all the active nodes. At the beginning of the object detection process there is the pcl-message, recorded by the depth camera described in Section 4.2. The pcl message is published by the camera and subscribed by the OM-N.

5.1 Creation of 3D bounding boxes

The transformation of the 2D bounding box from *YOLO* into a 3D bounding box is conducted with the object mapping node (OM-N).

This node is constantly subscribing to the topic of the 3D pcl message created by the camera as displayed in Figure 4.1. The pcl message consists of points that are created by combining the color information of the pixels from the RGB camera and the depth information of the depth camera. These points have a color value and a x,y,z value and all the points of one pcl message are called colored pcl. This triggers the cloud callback function of the subscriber. Within the callback function the highest index of the mongoDB is received from the viz-N for the correct identification of messages. The index number, equal to the number of elements stored within the mongoDB, will be increased by one and given as identification number for the stored message sent to the db-N. The points detected by the camera can only be located as relative positions to the camera frame and have to be transformed into the map frame in order to be able to locate them as absolute positions in the map. This is done by the transform function message, also received by the OM-N. This message holds the translatory and rotatory information needed to transform points from the camera frame into the map frame as shown in Figure 4.7b.

5.1.1 Bounding box

For *YOLO* a 2D RGB image is needed, therefore the RGB image of the Asus camera is used. This image is sent from the OM-N to *YOLO* node and the

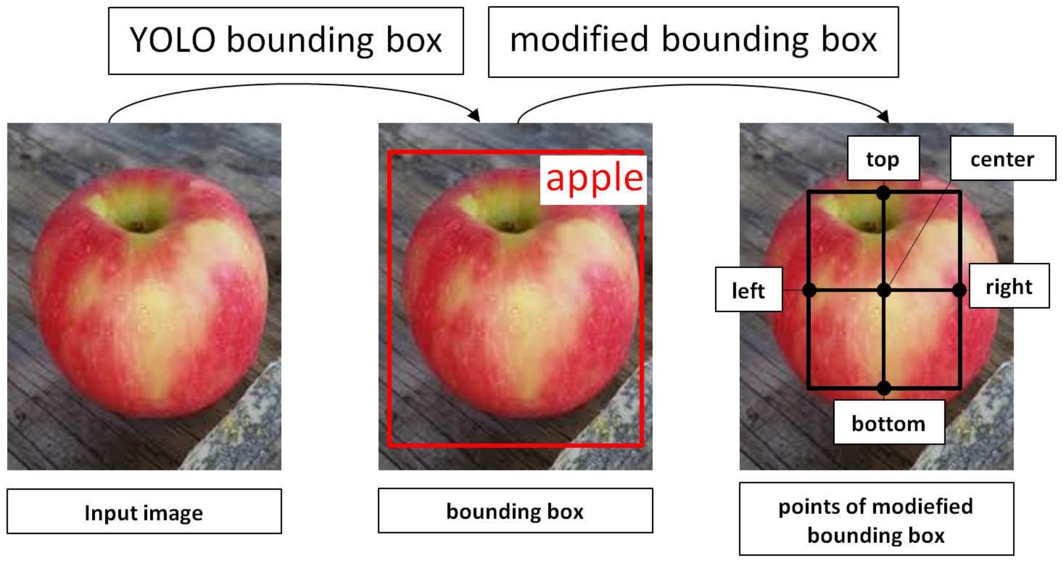
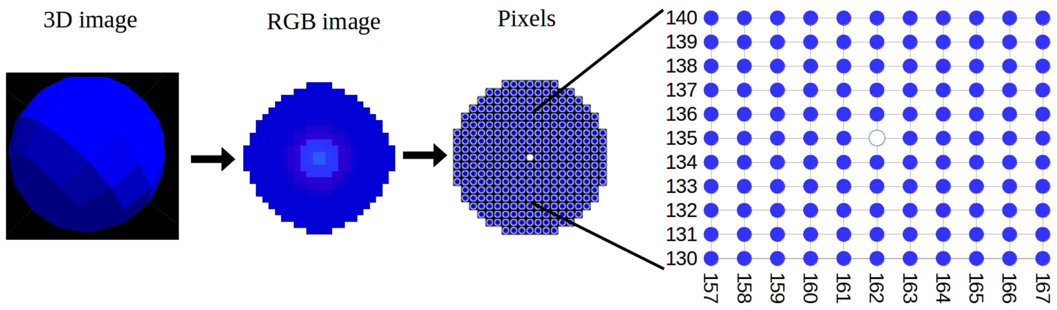


Figure 5.1: Modfication of bounding box

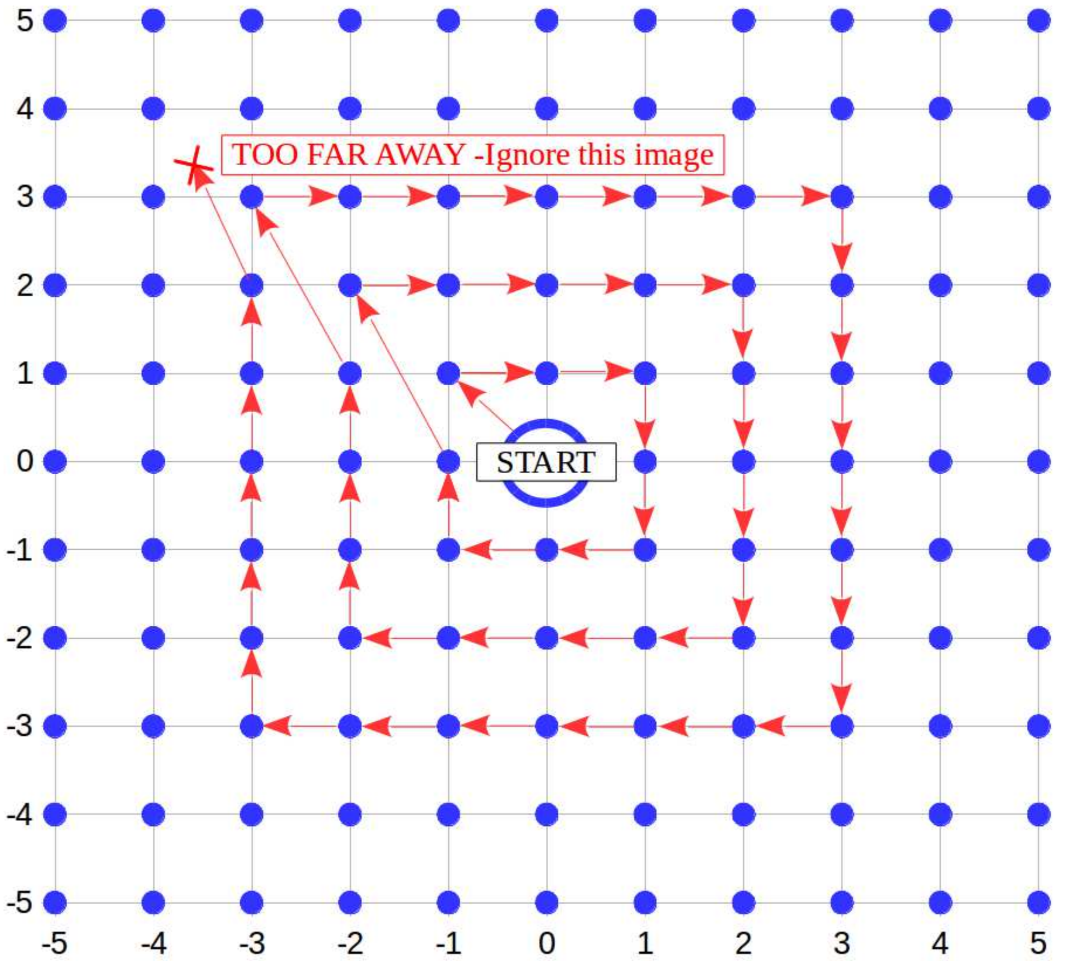
OM-N stops computing. *YOLO* uses the 2D image as described in Chapter 4.3 and publishes the resulting *bounding_boxes* message. The OM-N continues computing once it receives the *bounding_boxes* message.

Every *bounding_boxes* message from *YOLO* includes one bounding box for every detected object. The center point, width and height of the bounding box is used to create the 3D bounding box. This is done by firstly using the center point of the 2D bounding box within the 2D image to define the start point. The position of the center point within the image is defined as a x and a y value which is the same x and y value for the pcl. Since it is possible that the pcl may not exist at this x and y position a nearest point search has to be conducted.

A non existing pcl value is defined as a *NaN* value which can be displayed by examining the pcl at the relevant [x,y] position. *NaN* values are created because the depth information cannot be obtained. This can happen due to shiny, transparent, very matt or absorbing surfaces [18]. If the center point is a *NaN* value the nearest point search starts from the closest point north-west of the center point as displayed in Figure 5.2b, which is the defined as the first layer, and seeks for *NaN* values. The search for *NaN* values is displayed in Figure 5.2b. The points in Figure 5.2b are representative for the pixels of the detected image and are created as displayed in Figure 5.2a. Every pixel includes, additionally to the RGB values displayed in Figure 5.2a, the depth information which is detected by the ASUS depth camera. From the START point the search for *NaN* values continues clockwise, rotating towards the



(a) Detected image pixels displayed in form of points



(b) Direction of nearest point search without NaN value

Figure 5.2: Nearest point search for removing NaN values

beginning point. If the search would result in the start point of the first layer, north-west of the START point, the search continues by adding one more layer and therefore move one more point away from the START point, which results in the point north west of the starting point of the first layer. For each layer the point moves further away from the center point. The search is limited by a number of maximum layers which can be changed and the search is displayed in Figure 5.2.If the search would exceed the number of maximum layers the processed image and pcl is ignored. If any point within the search, including the start point, is properly defined, meaning there is no *NaN* value at the [x,y] position of the pcl, a new bounding box is created.

The newly created modified bounding box consists of five points shaped as a cross. This cross has a horizontal and a vertical line with the center point in middle as displayed in the third image of Figure 5.1. The length and width of the newly created bounding box are decreased. The proportions of the length and width still remain the same as the proportions of the length and width defined by the *bounding_boxes* message of $YOLO$. It is created as a cross, because the 4 points defining the cross are more likely belong to the same object as the center point, rather than the four corners of any bounding box. If the four points defining the bounding box would be chosen as corner points for the reduced bounding box displayed in the third image of Figure 5.1, these chosen points would just barely on the apple. As displayed in Figure 5.1 the points defining the cross are already on the apple even for increased length and width of the bounding box. This proves that especially for spherical shaped objects the four points defining the bounding box should be created as a cross. The same assumptions can be made for basic objects that are shaped as cylinder. This is especially the case for bottles with a slim bottleneck where the cross is perfectly fitted to the bottle while the corners of the bounding box are not part of the object even for a big reduction of the length and width of the bounding box.

5.1.2 Transformation of modified bounding box

All the four chosen points, same as the center point, are tested for *NaN* values. If these four points are properly defined in the pcl the measuring of the bounding box in 3D space can begin. The definition of the length and width must not be in pixels any more, since this is no viable measurement in 3D space, therefore the float values representing distances are used. In order to determine the distances in 3D space the 5 points have to be transformed into the map frame firstly by using the previously defined transform function. The frame transformation from the camera to the map is displayed in Figure 4.7b and the points that are transformed are displayed in Figure 5.3. The height is

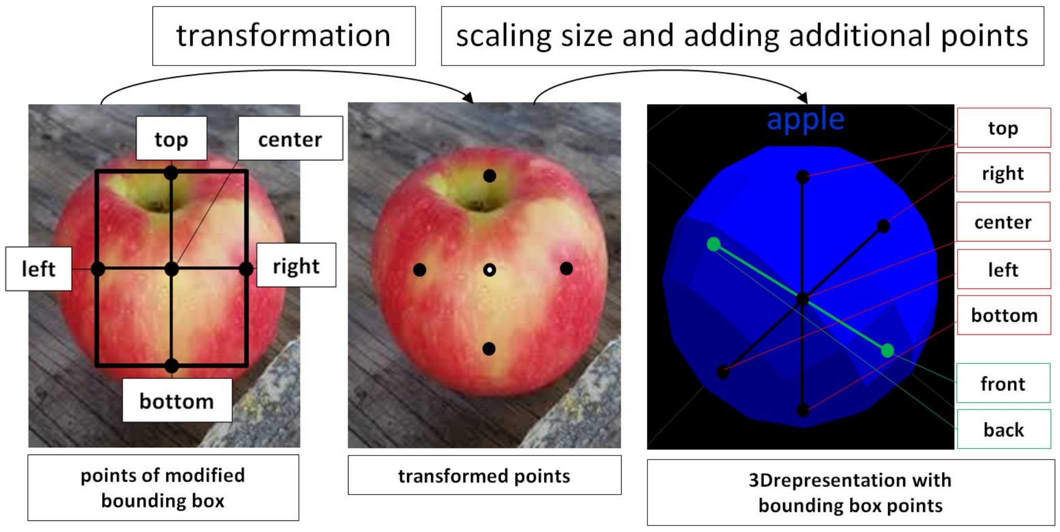

transformation
scaling size and adding additional points
top
center
left
right
bottom
points of modified bounding box
transformed points
apple
top
right
center
left
bottom
front
back
3Drepresentation with bounding box points

Figure 5.3: Transformation of bounding box for the creation of modified bounding boxes, source: [52]

defined as the difference between the z value of the upper point of the cross and the z value of the lower point of the cross. If only the difference of the x, or y values are used for calculating the width of an object the value of the width could be 0. This happens because the orientation of the object might be exactly in the direction of one axis and the measured x or y values might then be 0 if the camera looks at the object directly from one side, in the direction of one axis. Therefore the width has to be calculated as an *Eulerdistance* between the two y and the two x values of the left and right point of the cross. Figure 5.3 shows the transformed cross with the new length and width and displays the transformed points of the bounding box. The additional two points are added with the same distance as the distance between the left and right point.

If the height is smaller than the width by a factor, that is set by the user, the object is presumed to be a table or lying on some level and therefore treated differently than a standing object. Assuming the object is standing the height and the width of the bounding box is already defined. Since the 3rd dimension of the object cannot be assumed correctly for only one viewpoint, two additional points are added to the five original points. The additional points displayed in the third image of Figure 5.3 and the representation of the apple is shown as a blue sphere, visualized in RVIZ, with the same size as the apple.

The center point from the initially created bounding box in 3D would appear as part of the surface, or at least close to it, therefore it has to be moved in the looking direction of the camera by the value of half the width of the object. The third image of Figure 5.3 shows the new center point which is inside the object and the old center point, marked with a white point in the middle, which would be part of the object if not moved.

For the lying object the same presumptions cannot be made, since moving the center point in the looking direction of the camera by half the width and creating a cuboid, would result in a false 3D representation of the object. Therefore the center point is not moved and the height of the object is set to a fixed value for displaying it properly. This fixed value of the height is used in the viz-N in Section 5.3 for further computing. The width in x and in y is computed the same way as the standing object. With the position of the center point, the height and the width the 3D bounding box is defined.

The 5 points of the 3D bounding box as well as the label and the id are used to create a custom messages, which is the representation of one object. An array of custom messages is packed into one message and sent to the db-N after processing all bounding boxes within the *bounding_boxes* message. There is also the possibility that a bounding box is not used since *NaN* values appeared during its transformation and thresholds are exceeded. In this case the object is overwritten by the next processed object and the message sent to the db-N only includes the properly defined 3D bounding boxes.

5.2 Data storage

Before it is possible to store messages into the *mongoDB* the database server has to be stated. All the messages stored will be located in the defined path and can be accessed from there. For every new environment a new database has to be created, since it would make no sense to simply place objects detected in one environment into another.

The message sent from the OM-N is recieved by the db-N and the callback routine is started. The callback routine uses the received data with the type of the incoming message and inserts it into the database with a command provided by *mongoDB*. The stored message is displayed in Figure 5.4.

The *persistent object mapping (pom)* array includes the message id and an array of stored *pom* messages and the standard header file of *ROS* with the time stamp the frame id and the sequence number. Every *pom* message within this message array has the information of one 3D bounding box representing an object. This includes the height, width, length, and the coordinates of the center point of the 3D bounding box. Furthermore the id of the element, the label of the 3D bounding box and another standard *ROS Header* is defined.

```
##pom_Array
    #standard ROS header
        Header header
    #message id
        int32 message_id
    #stored messages
        pom_msg[] msgs

##pom_msg
    #standard ROS header
        Header header
    #id of the element
        uint32 element_id
    #label with name and confidence from YOLO
        tmc_vision_msgs/Label label
    #coordinates of center point
        float64 x
        float64 y
        float64 z
    #length width and height of 3D bounding box
        float64 lx
        float64 ly
        float64 lz

##tmc_vision_msgs/Label label
    #name of the object
        string name
    #confidence for the detected object
        float64 confidence
```

Figure 5.4: Message stored in the mongoDB

5.3 Data management and visualization

The *mongoDB* holds all the 3D bounding boxes of the detected objects and a query has to be conducted to search for these bounding boxes. The gathered data is filtered with some thresholds, multiple bounding boxes surrounding the same object are replaced and shape estimations based on the name of the object are made and the resulting modified bounding boxes are visualized. This bridge between the stored bounding boxes in the *mongoDB* and the visualization of these bounding boxes with RVIZ is the viz-N.

The result of the *mongoDB* query, conducted in the viz-N, is an array of messages. This array holds all stored messages within the *mongoDB*. The number of stored messages is equal to the maximum id of the *mongoDB* which is sent to the OM-N, after every conducted query, which is received by the OM-N at the start of the callback routine explained in Section 5.1. Since the viz-N has to compute all the messages from the beginning, it starts by processing the first stored message, which is the oldest one and continues until it has reached the newest message.

5.3.1 Modified bounding box

The bounding boxes in this Chapter are without exceptions 3D bounding boxes and therefore simply named bounding boxes. A *vizArray* is created to store the gathered data from the mongoDB in form of modfied bounding boxes. The *vizArray* consist of a number of modified bounding boxes that are defined by their geometric shape, position and name. Every message in the mongoDB has stored at least one bounding box. A for-loop runs through all the stored bounding boxes within the messages and creates modified bounding boxes. Every new modified bounding box is filtered by thresholds, shape estimations are made and the modified bounding box can replace other modified bounding boxes within the *vizArray*. If the new modified bounding box does replace any other modified bounding boxes the entries of the replaced modified bounding boxes are removed from the *vizArray*. If it does not replace any other modified bounding box a new entry in the *vizArray* is added.

The information of the name, coming from the message, is instantly stored into the newly created modified bounding box and will not be changed, but the modified bounding box might not be saved into the *vizArray*. One of the reasons why this modified bounding box might not be stored into the *vizArray* is that persons are not displayed, since they are usually moving fast without a persistent position and therefore not suitable for a persistent object map. Another possibility is that the classification confidence is below the confidence threshold. The classification confidence is the confidence that the object was detected correctly and is coming from *YOLO* and stored through the OM-N and the db-N to the *mongoDB*. The confidence threshold is a fixed number and is set for the results shown in Chapter 6. Every detected object with a detection confidence lower than the threshold will neither be stored as modified bounding box in the *vizArray*, nor processed by the viz-N.

5.3.2 Replacing a modified bounding box

In order for a modified bounding box, of the currently processed object, to replace an existing modified bounding box within the *vizArray*, the distance between their center points has to be below a threshold. The threshold is set to half the length of the currently processed object. This ensures that only modified bounding boxes of the *vizArray* within the area of the new modified bounding box are replaced by the new modified bounding box. Every modified bounding box in the *vizArray* has a weight factor that determines the reliability of the stored location of the modified bounding box. If a modified bounding box of the *vizArray* is replaced the weight factor of this new modified bounding box within the *vizArray* is a sum of the replaced modified bounding boxes' weight factor added to the new modified bounding boxes' weight factor because the reliability that the location of the modified bounding box is correct has increased. The reliability increases because when one modified bounding box replaces another modified bounding box, the result is assumed to be an object detected in multiple scenes viewed by the camera. Therefore the confidence that the position and size of the object is correct increases with every new replacement and therefore the weight factor is increased. The factor with which the weight factor is increased can be modfied and is set in Chapter 6.

When replacing the old modified bounding box, the position of the old modified bounding box is adjusted towards the position of the new modified bounding box and the length and width is adjusted towards the length and width of the new modified bounding boxes The adjustment only results in a mean value of the two modified bounding boxes' length, width and position, if both modified bounding boxes have the same weight factor. Otherwise the adjustment occurs by factor smaller, equal to the weight factor difference of the two modified bounding boxes. If the new modified bounding boxes distance to all the modified bounding boxes of the *vizArray* is measured and the possible replacements occurred the modified bounding box is stored into the *vizArray* with its position, size, shape, name, location and id.

5.3.3 Shape estimation

While creating a new modified bounding box the name of the detected object resulting in the creating of the modified bounding box is checked for indications that could lead to the shape of the object. For example the apple, orange, or ball will most likely be shaped as a sphere or at least close to it. Therefore the 3D representation of these objects should also be a sphere instead of a cuboid. If the name is one of the mentioned ones the shape of the object is set to *SPHERE*. If the name equals cup, bottle or vase the shape of the modified

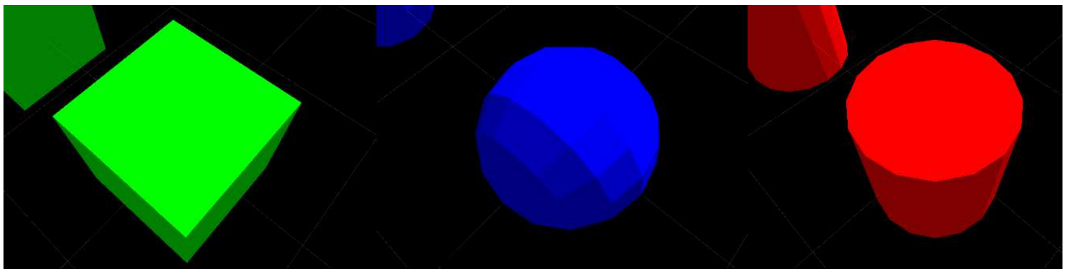


Figure 5.5: Basic markers displayed with RVIZ

bounding box is, with the same reasoning, set to *CYLINDER*. For all the other objects the shape is set to *CUBE*.

The length and width of the created modified bounding boxes are slightly changed by a constant factor depending on the shape of the modified bounding boxes and left unchanged for cuboids, except if the modified bounding box replaces another. If there is a modified bounding box within a certain distance to the currently created modified bounding box within the *vizArray* this modified bounding box will be replaced with the new modified bounding box and the length, width as well as the position is newly calculated.

5.3.4 Marker creation

After all the bounding boxes of the currently processed message have been computed, the *vizArray* is transformed into a *markerArray*, published and can be viszualized by *RVIZ*. The *markerArray* has stored the same information of the bounding boxes as the *vizArray* with the only difference that the stored information is stored in form of markers and a color is added. Markers are objects with a size, shape, name, color and location used by the program *RVIZ* and some example markers are displayed in Figure 5.5. The transformation of the *vizArray* to the *markerArray* is conducted by entering the parameters from the *vizArray* into the *markerArray* accordingly and adding color information. The colors of the markers are set in Chapter 6. If all messages, stored in the *mongoDB*, are processed through with the *vizArray*, the *markerArray* is constantly published.

6 Experiments

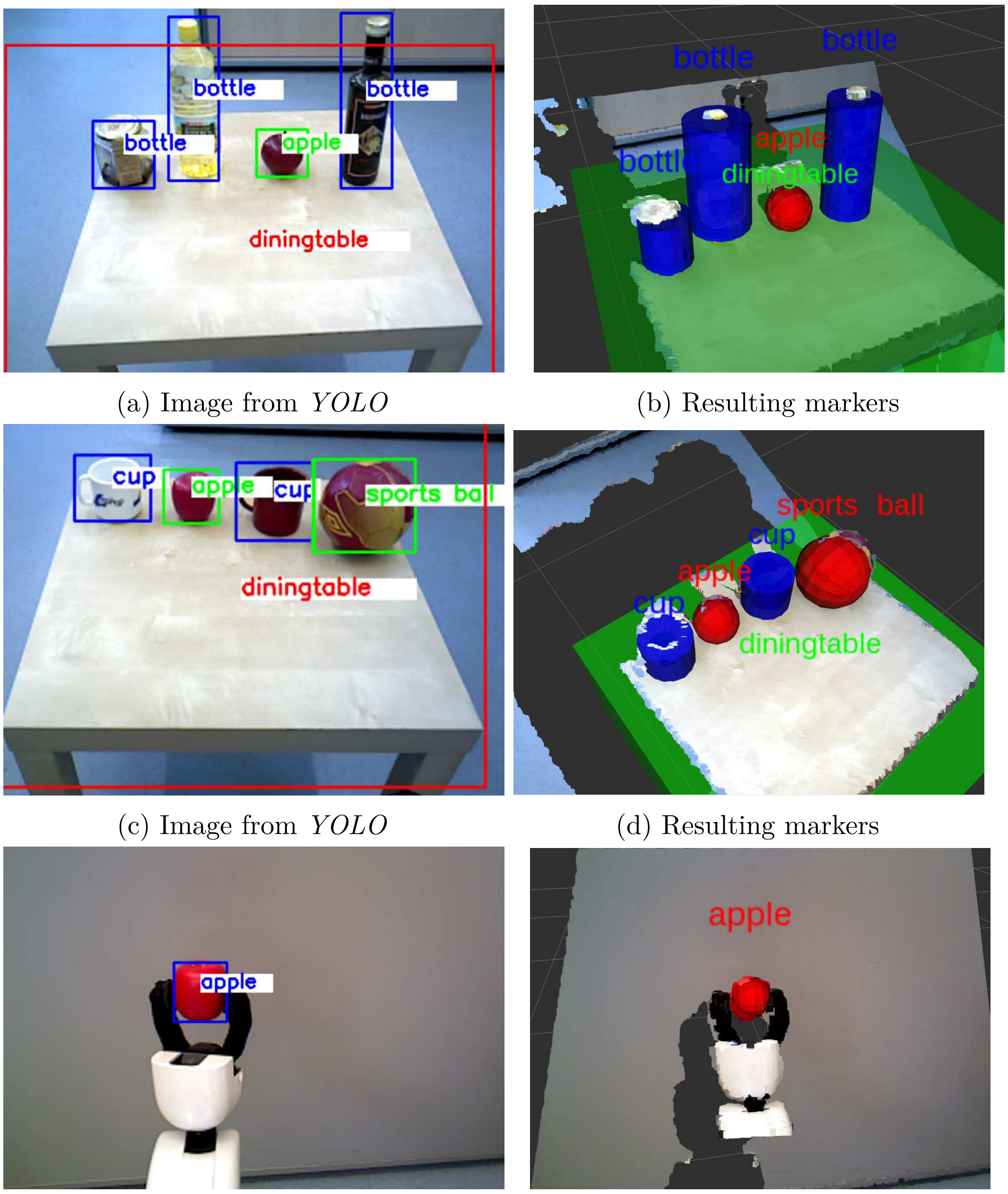
This Chapter describes the conducted experiments for testing the implemented algorithms. In the first experiment the a static environment is used to determine the correct estimation and visualization of the 3D bounding boxes surrounding the objects. In the second experiment an object is moved and the resulting marker after the movement is displayed. This test is conducted to show the correct replacement of objects when they are detected in multiple locations. The third experiment is conducted while the robot is moving and viewing the same area of objects. In this experiment the multiple views of the objects should result in only one visualized 3D bounding box per object. The fourth experiment is conducted to show the persistency of the stored objects by restarting the robot and visualizing the same objects before and after the restart. In the last Section the durations of the implemented modules and programs are measured by processing 100 different scenes captured by the robot.

For conducting the experiments the robot and the additional Computer are used as displayed in Figure 3.1 and explained in Chapter 3.1. *RVIZ* is used on the additional Computer to display the markers published by viz-N and the pcl published by the camera of the robot and an additional image visualization program from *ROS* is used to display 2D images.

The parameters of the programmed nodes described in Section 5 have to be set for the correct visualization of the 3D bounding boxes. Colors of the markers, explained in Section 5.3 are set to green for cuboids, red for spheres and blue for cylinders. The factor for determining tables and other lying objects as described in Section 5.1 is set to seven and the confidence threshold for *YOLO* is set to 35%. The maximum layers for the NaN search described in Section 5.1 is set to three.

6.1 Static environment

In the first experiment the robot remains at one position and all the objects do not move. As displayed in Figure 3.1 the RGB-D camera of the robot sends the pcl image to the OM-N and the 2D RGB image of the camera is sent to *YOLO*. The images displayed in Figure 6.1a, 6.1c and 6.1e are the combination of the resulting bounding boxes message from *YOLO* and the 2D image coming



(a) Image from *YOLO* (b) Resulting markers

(c) Image from *YOLO* (d) Resulting markers

(e) Image from *YOLO* (f) Resulting marker 3

Figure 6.1: Markers with static robot

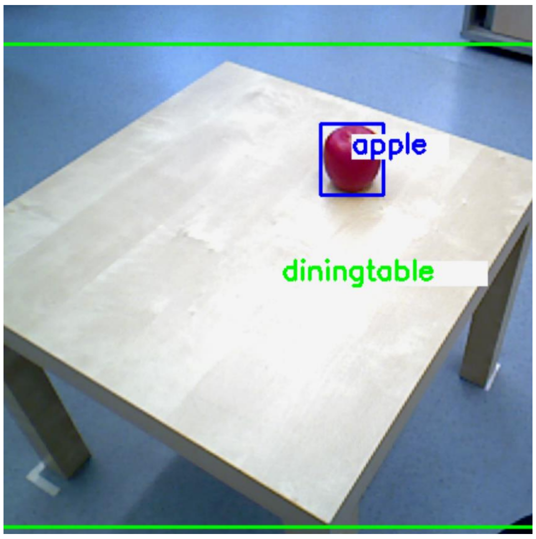
from the OM-N. They show the 2D image, as detected by the RGB-image of the depth camera and the detected objects, which are surrounded by their corresponding bounding boxes and names. The diningtable is surrounded by a bounding box, but this bounding box is not suitable and will create a representation of the diningtable, which is larger than the real diningtable. Apart from this error all the present objects have been detected and will be transformed into 3D bounding boxes.

As described in Chapter 4 and displayed in Figure 3.1 the message created by the OM-N is sent to the db-N which stores it into the *mongoDB*. The viz-N creates the corresponding markers out of the stored objects and sends them to *RVIZ* which can display the pcl and the created 3D representations of the detected objects. These representations and the detected pcl of the depth camera are visualized in Figure 6.1b, 6.1d and 6.1f. With a small margin of error the markers fit the bodies of the objects. As predicted the size of the diningtable is too big, but the marker captures the form of the table correctly which can be seen in Figure 6.1b. The presented markers are persistent, which means that they will remain in the same position even after the camera looks in a different direction or the robot is restarted.

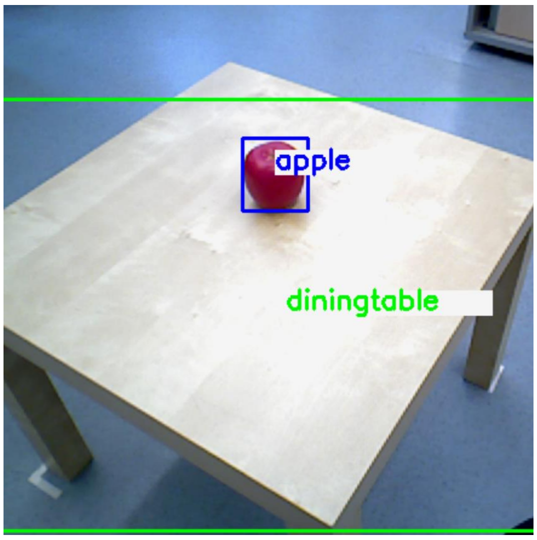
To demonstrate the possibility of detecting object which are not only lying on a table, the persistent object mapping algorithm is tested on an object that is grasped by the HSR. The input image form *YOLO* is displayed in Figure 6.1e and the result is displayed in Figure 6.1f. This test shows that the presented algorithm can also detect objects that are grasped by the HSR.

6.2 Relocating objects

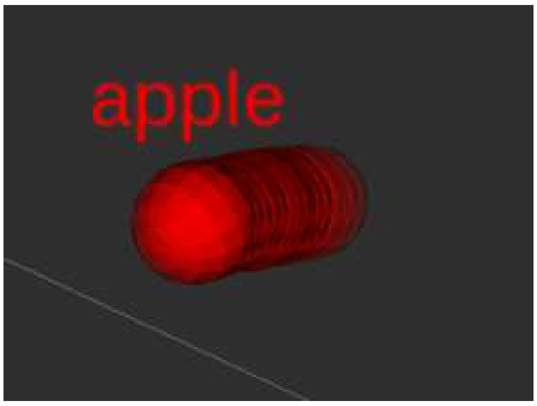
Now that the static representations of the objects have been tested and displayed in Figure 6.1, the next test is conducted to see if moving objects are also monitored correctly. This is shown by detecting and documenting the movement of an object which is moved by hand. For the results displayed in Figure 6.2 the moving object is an apple, which has been selected because of the high detection confidence from *YOLO*. Figures 6.2a and 6.2b display the apple, detected by *YOLO*, in different positions. The apple has been moved from the position displayed in Figure 6.2a to the position displayed in Figure 6.2b. During the time of the movement the objects mapping process, as displayed in Figure 3.1, constantly stores the detected objects into the *mongoDB*. The representation of the object, which is detected latest, overwrites, as described in Section 5.3, the former representation detected at a different time, which results in a moving marker that follows the movement of the apple. Since the viz-N only shows the resulting marker, which is the one detected latest, the



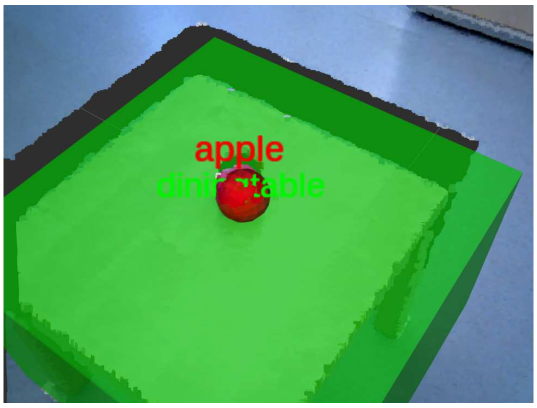
(a) Starting position



(b) End position



(c) Movement of apple



(d) Resulting markers without pcl

Figure 6.2: Moving object

code has to be changed in order to display the marker at different times. This is done by changing the display time of the marker to ten seconds and visualizing all the objects stored within the *mongoDB*, which results in the visualization of the marker during the movement of the apple for ten seconds each. Since the viz-N takes about 50ms for displaying one marker the time taken for displaying all the objects stored is negligible to the ten seconds displaying time. The result of this setting is displayed in Figure 6.2c. The diningtable has not been moved, the pcl is not necessary and the names would not be readable, therefore they are not visualized in Figure 6.2c. The result displayed in Figure 6.2c shows that the marker followed the movement of the apple correctly from the starting to the end position. The resulting markers after the movement and without the prolonged display time is displayed in Figure 6.2d.

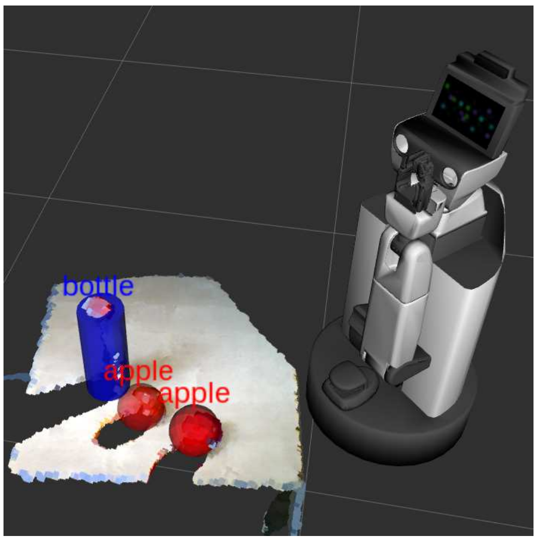
6.3 Relocating robot

In the next test the robot is moved while looking at a diningtable with three objects placed on it, an apple an orange and a bottle. The movement of the robot is displayed in Figure 6.3 and the robot moves from the position displayed in Figure 6.3a to the position displayed in Figure 6.3d. Figure 6.4 shows the movement process of the robot, including the markers and the pcl at different times of the movement.

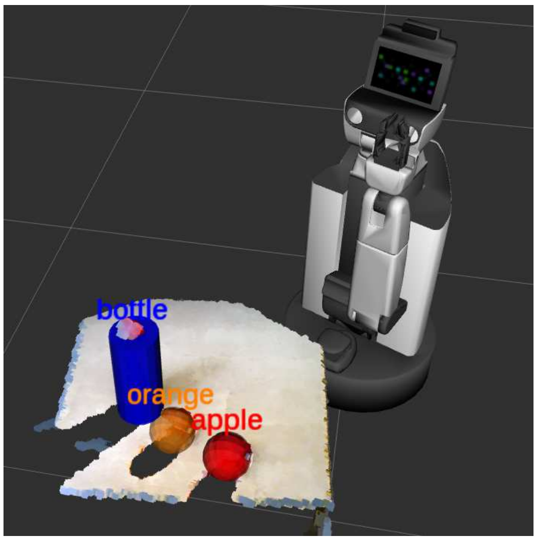
Using the same settings of the code as it was used in the second experiment, with the moving apple in Figure 6.2c, the movement of the markers is displayed in Figure 6.3c. The result should ideally be only one marker that does not move, but there is a problem with the positioning and orientation of the robot in the room. The error caused by the repositioning and reorientation of the robot is displayed in Figure 6.3c. The result in Figure 6.3d shows, that the algorithm of the viz-N can adjust the positions of the markers even if the robot moves and reorients itself during the movement. The resulting markers are correctly positioned, with a small margin of error, which can be seen in Figure 6.3d. During the movement of the robot the orange was wrongfully detected as apple in the first scene but was then detected correctly. This is due to the detection program *YOLO* that makes slight mistakes when detecting objects.

6.4 Persistence

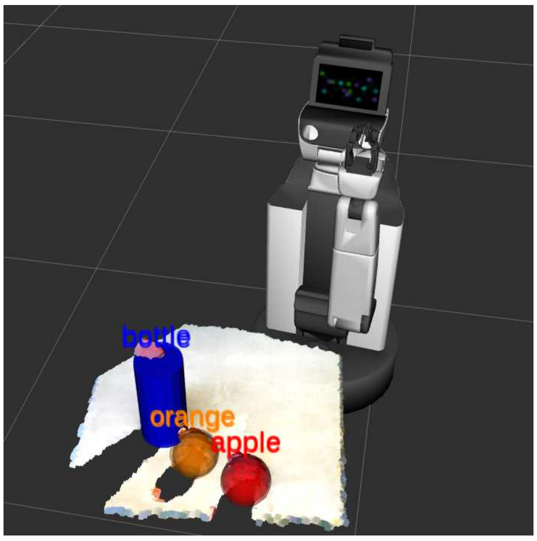
This experiment shows the detections of the robot before and after a restart of the robot. These detections are stored and all detections combined are displayed as 3D representations in Figure 6.5a.



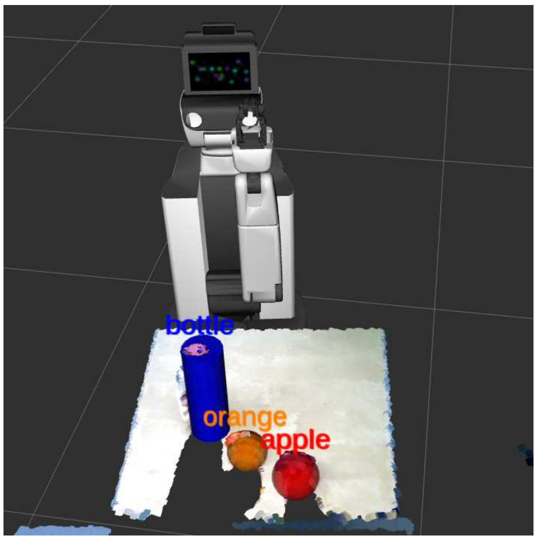
(a) Position one



(b) Position two

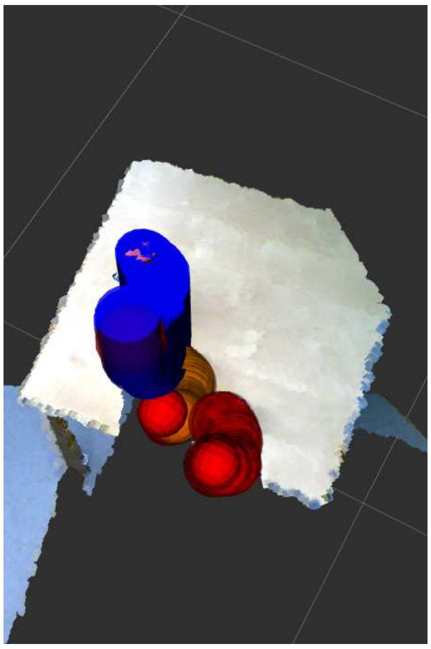


(c) Position three

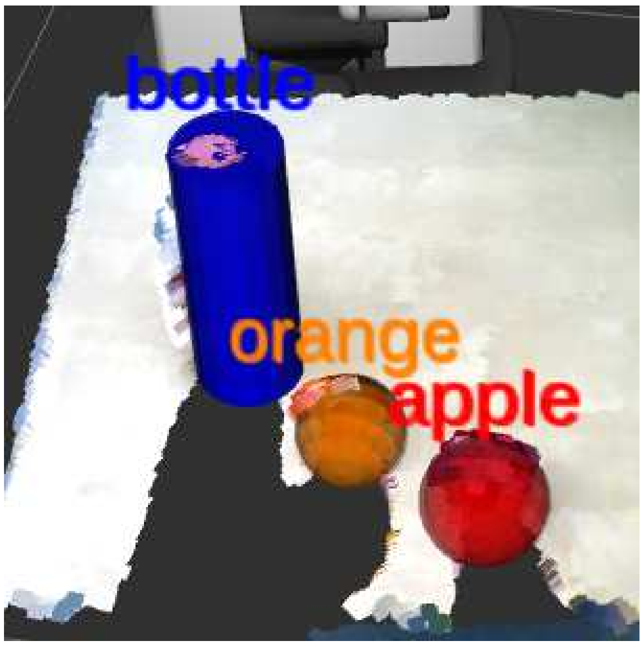


(d) Position four

Figure 6.3: Moving robot



(a) Markers at different times



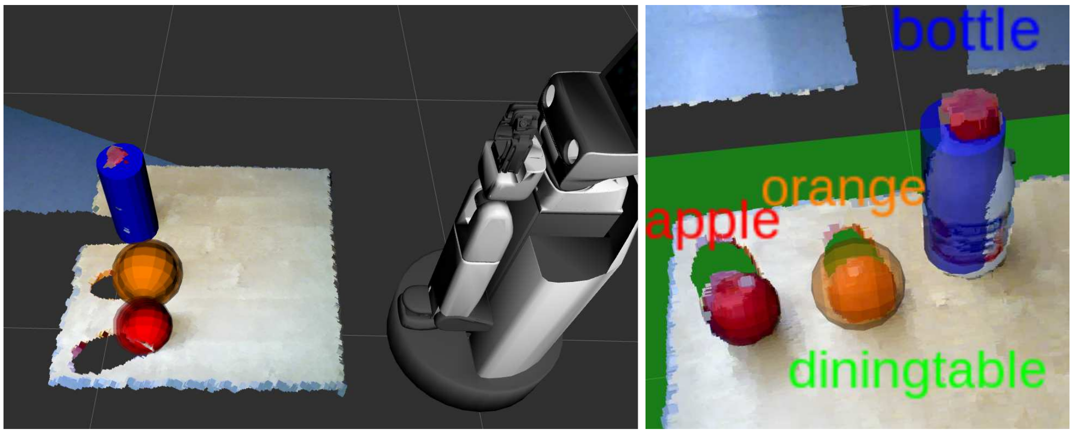
(b) Resulting markers

Figure 6.4: Movement of robot

After the robot has detected all present objects and stored them into the *mongoDB* the robot can reload its battery and shut down. This is tested by detecting some objects, shutting the robot down, restart it again and viewing the same objects. In Figure 6.5a all stored objects before and after the restart are displayed together. These stored objects are transformed into markers by the viz-N explained in Section 5.3 and the visualized objects are represented by their corresponding markers in Figure 6.5a. The final result, after the restart and the processing of the viz-N, is displayed in Figure 6.5b. As it is shown in Figure 6.5, the robot can be restarted and will still detect the same objects at the correct positions as before the restart. The conducted experiment only works when the reorientation process of the robot after the restart is conducted within a margin of error, meaning that the robot has to locate and orient itself correctly in space to detect the objects at the same positions.

6.5 System analysis

In this Section the durations of the implemented modules and programs are measured by processing 100 different images from the robot. The durations of the single processes and nodes is displayed in form of two diagrams. The first



(a) Markers before and after restart (b) Final markers

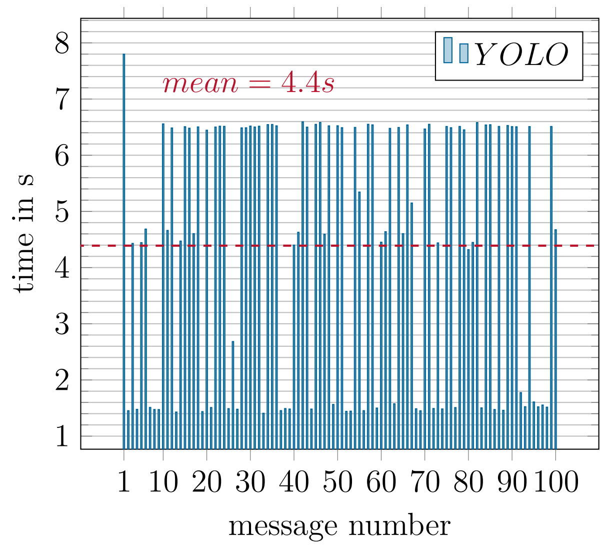
Figure 6.5: Restart of robot

diagram shows the measured durations for every single image and the second diagram displays an overview of the measured durations in form of a histogram.

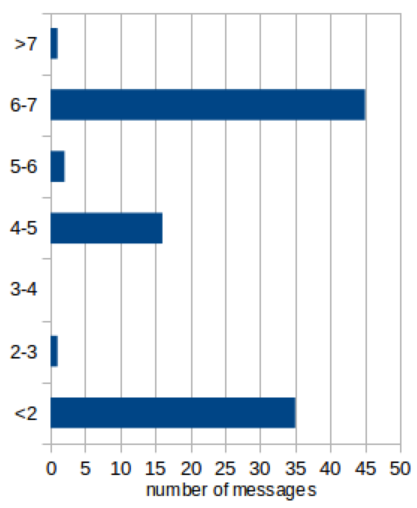
Figure 6.6 shows the durations of the OM-N that creates the 3D bounding boxes. It is separated into the durations of *YOLO* in Figure 6.6a, 6.6b and the whole process of the OM-N in Figure 6.6c, 6.6d. As it can be shown the process for detecting objects with the program *YOLO* running on the *Jetson TK1* of the robot takes up to 8s to detect objects in a processed image. Although the mean value of the durations for detecting objects is 4.4s, *YOLO* could run 200 times faster with a speed of 50 frames per second if the necessary hardware is provided. Therefore the implemented algorithm can only detect new objects with a delay of at least 1.67 seconds, which is the fastest duration for *YOLO*. The additional mean value of 330 ms added to the 4.4 s from *YOLO* is due to the transformation of the pcl message and the transportation of the information via the wifi router, because only after the successful storage of the processed data the OM-N can start to process new data.

The durations of the db-N are displayed in Figure 6.7. These durations represent the time taken for storing the newly processed bounding boxes from the OM-N into the *mongoDB*. The more data are stored in the database the more time is taken to store new data. The duration of the db-N is growing but still neglectable compared to the up to 8s duration of *YOLO*.

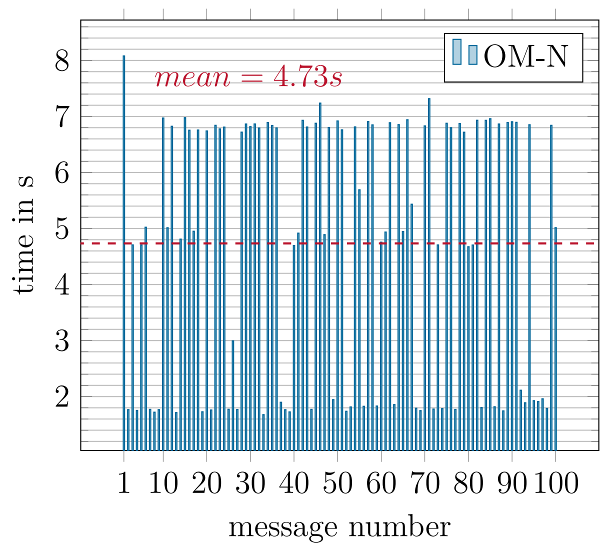
Figure 6.8 shows the durations of the viz-N that processes and publishes 3D bounding boxes stored in the mongoDB. The average duration of 50 ms for the viz-N, same as the duration of the db-N, is neglectable compared to *YOLO*.



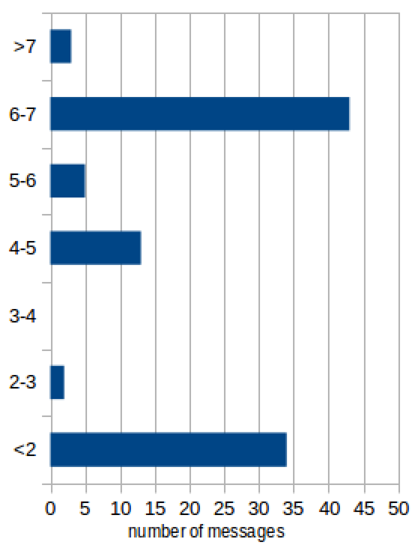
(a) Durations of YOLO



(b) Histogram of the durations

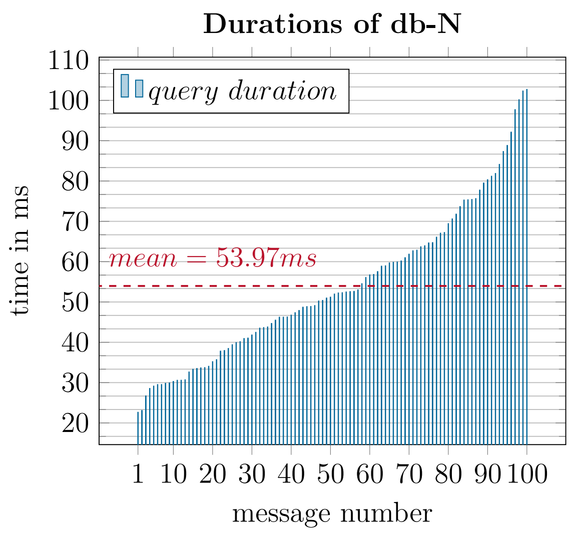


(c) Durations of OM-N

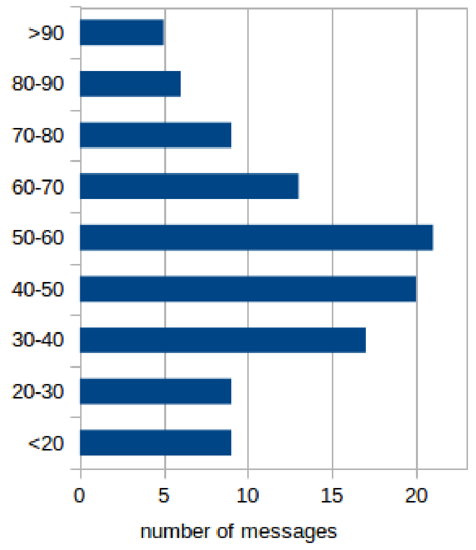


(d) Histogram of the durations

Figure 6.6: Durations of YOLO and whole OM-N for 100 messages

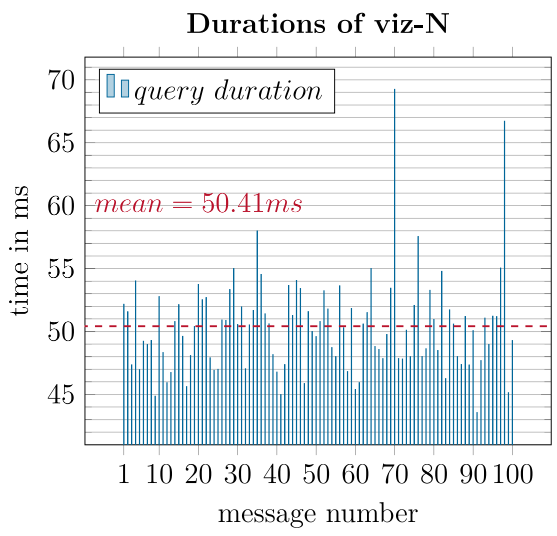


(a) Durations

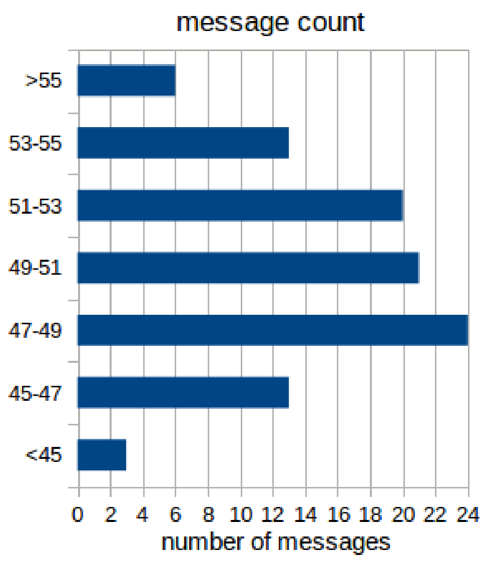


(b) Histogram of the durations

Figure 6.7: Measured durations of db-N with histogram for 100 messages



(a) Durations



(b) Histogram of the durations

Figure 6.8: Measured durations of viz-N with histogram for 100 messages

7 Conclusion

Development of robots interacting with humans is a vast growing sector in mobile robotics. In order to integrate robots in human environments, detecting objects and estimating their location and size and saving the gathered information for further processing is a vital contribution.

In this thesis a process is presented that detects objects, creates 3D bounding boxes surrounding the detected objects and stores, filters and visualizes the 3D bounding boxes. This process creates a persistent object map, a virtual environment where the robot can access information about detected objects and the calculation of absolute or relative distances can be conducted. By using state-of-the-art object recognition algorithms, databases and sensors the created persistent object map can detect and permanently store objects. This process can improve the trajectory planning process of the robot towards objects in order to interact with people more efficiently. The creation of 3D bounding boxes out of 2D bounding boxes is made possible with the presented process and these 3D bounding boxes are permanently stored in a database where they can be accessed even after the restart of the robot or supporting hardware. The locating process of requested objects is improved because the positions of detected objects are already stored in the persistent object map and can be accessed at any time. Also relative positions to other objects or humans can be calculated by using the positions of the objects stored in the persistent object map.

7.1 Supporting programs

The Robot Operating System Framework is used for the communication and implementation of the process in form of nodes. Even if the nodes are not running in the same operating system, or on the same hardware, the framework can establish the connection and communication of the nodes. The *HSR* from Toyota executes the implemented nodes and provides the necessary hardware to accomplish the tasks of object detection and image processing. One of the currently fastest detection algorithms, YOLO, is implemented to detect objects in the images recorded by the camera of the HSR from Toyota. Representations of objects are stored in the database *mongoDB* even after the camera of the robot does not view the objects any more. After a restart of the robot the information of the detected objects is still present and the representations of the objects can be reloaded.

7.2 Results

The conducted experiments present the persistent object mapping process implemented on the Computer of the HSR and a second computer. The results show the correct detection of objects and creation of 3D bounding boxes surrounding the detected objects. The experiments demonstrate that the algorithm can replace multiple views of the same object with one resulting 3D bounding box after and during the movement of objects or the robot. The experiments also show that detected objects are permanently stored in a database and their correct locations in space, and additional information, can be accessed even after the restart of the robot. Time measurement tests demonstrate that the duration of the persistent object mapping process without the duration of the detection program takes only 330ms in average process one image. The process filters and visualizes new detected objects within less than 70 ms.

All the experiments conclude that the implemented persistent object mapping process can be used in static and slowly chaining environments for locating and storing 3D representations of entities present in the working area of the robot. The stored data includes shapes, sizes and locations of the detected objects and further information can be added to the stored objects.

7.3 Limitations and future work

There are some limits mostly due to the timing problem of *YOLO* and the reorientation and relocation problem of the robot. The object detection algorithm running on the Jetson TK1 GPU takes up to 8s to detect objects in the input RGB image. Due to this fact the positioning and location of objects is difficult to capture. Especially at fast movements of either the robot, or any object, the object detection algorithm is not able to locate objects precisely any more and creates multiple instances of markers representing the same object. This could be solved by running the detection algorithm on the PC. Unfortunately running YOLO on the PC would defy the purpose of the camera on this robot, because the detected pcl image is very large and should not be transported, but rather processed directly on the robot. Due to the speed limitations of the network of the used router the transportation of the pcl would consume most of the bandwidth and the detection process would be slowed down rather than accelerated. This could be improved by using better hardware. Either the GPU can be replaced or the network or both which would tremendously increase the detection speed. YOLO is capable of detecting objects with more than 50 frames per seconds using sufficiently powerful hardware.

Even though the intended scenario for the persistent object mapping process only considers static or slowly moving environments where the robot chooses when to detect objects we would like to consider real dynamic environments where objects and the robot can be moved and the detection process runs in real time. The persistent object mapping algorithm provided with the mongoDB does not delete any objects in the database but rather fuses the detections stored in the mongoDB into markers. Due to this fact the database can grow very large and if objects are replaced the algorithm still keeps the remnant of the representational marker of the object. This could be solved by additional algorithms with the ability to detect space information of present shapes. If no entity is detected at the location where the object is assumed and the marker is placed, the marker could be removed. Also the robot could be moved during a restart and the relocation and reorientation process of the robot, which currently is erroneous, would also be improved.

7.4 Outlook

Beyond what has been demonstrated in this thesis there are a number of possible upgrades and usages of the described persistent object mapping process.

The presented algorithm is already partly implemented in a project that should enable the HSR to automatically grasp objects. The object grasping algorithm could be further improved by using the full persistent object map including the locations of all detected objects. Trajectorys towards objects have to be calculated and the persistent object map can help with the calculations. Positions of the objects are already stored in the persistent object map and trajectories to the most relevant objects could be calculated in advance when the robot is not used.

With the help of a octomap [45] or other algorithms that are used with pcls the orientations and shapes of some objects can be determined. This can work by locating points or occupied space within the existing 3D bounding boxes and then storing these points or the occupied space. The stored data can be continually expanded with every new detection from different angles and the orientation as well as a better model of the object can be obtained. Models of specific objects can be created with the help of 3D cameras. These models can be stored into a database of shapes and *YOLO* can be trained to detect these specific objects. These models can then be used instead of the basic shapes.

Objects stored in the persistent object map can improve the orientation and localization process of the robot. This is done by using the locations of objects as reference points in order to locate and orient the robot in the map. This can improve the localisation process of new detections which then again can be used to improve the localizing process of the robot. The more objects are viewed by the robot the better the localization process of the robot can be improved, under the condition that the objects have been localized correctly. The algorithm can, with some modifications, also be implemented on a different robot. If the robot has sufficient hardware specifications the persistent object map can be created and updated in real time.

Bibliography

[1] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrova, J. Young, J. Wyatt, D. Hebesberger, T. Kortner, *et al.*, „The strands project: Long-term autonomy in everyday environments,“ *IEEE Robotics & Automation Magazine*, vol. 24, no. 3, pp. 146–156, 2017.

[2] J. Forlizzi and C. DiSalvo, „Service robots in the domestic environment: a study of the roomba vacuum in the home,“ in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, ACM, 2006, pp. 258–265.

[3] „dish robot,“ in. [Online]. Available: `https://techcrunch.com/2009/10/16/panasonics-dish-washing-robot/`.

[4] T. Yamamoto, T. Nishino, H. Kajima, M. Ohta, and K. Ikeda, „Human Support Robot (HSR),“ in *ACM SIGGRAPH 2018 Emerging Technologies*, ser. SIGGRAPH ’18, Vancouver, British Columbia, Canada: ACM, 2018, 11:1–11:2, ISBN: 978-1-4503-5810-1. [Online]. Available: `http://doi.acm.org/10.1145/3214907.3233972`.

[5] A. M. Okamura, M. J. Mataric, and H. I. Christensen, „Medical and health-care robotics,“ *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 26–37, 2010.

[6] „Nvidia autonomous machines,“ in. [Online]. Available: `https://www.nvidia.com/de-de/autonomous-machines/robotics/`.

[7] „Amazon Robotics,“ in. [Online]. Available: `https://www.robotemi.com/`.

[8] N. Roy, G. Baltus, D. Fox, F. Gemperle, J. Goetz, T. Hirsch, D. Margaritis, M. Montemerlo, J. Pineau, J. Schulte, *et al.*, „Towards personal service robots for the elderly,“ in *Workshop on Interactive Robots and Entertainment (WIRE 2000)*, vol. 25, 2000, p. 184.

[9] „Pal Robotics Tiago Mobile Manipulator,“ in. [Online]. Available: `https://tiago.pal-robotics.com`.

[10] „Fetch robot,“ in. [Online]. Available: `https://fetchrobotics.com`.

[11] „pepper robot,“ in. [Online]. Available: `https://www.softbankrobotics.com/us/pepper`.

[12] „Robot operating System,“ in. [Online]. Available: `https://www.toyota-global.com/innovation/partner_robot/robot/images/mainimg01.jpg`.

[13] S. Thrun *et al.*, „Robotic mapping: A survey,“ *Exploring artificial intelligence in the new millennium*, vol. 1, no. 1-35, p. 1, 2002.

[14] D. Kragic, M. Vincze, *et al.*, „Vision for robotics,“ *Foundations and Trends® in Robotics*, vol. 1, no. 1, pp. 1–78, 2009.

[15] R. B. Rusu, „Semantic 3d object maps for everyday manipulation in human living environments,“ *KI-Künstliche Intelligenz*, vol. 24, no. 4, pp. 345–348, 2010.

[16] „Front Matter,“ in *Human and Machine Vision*, ser. Notes and Reports in Computer Science and Applied Mathematics, J. Beck, B. Hope, and A. Rosenfeld, Eds., Academic Press, 1983. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/B9780120843206500025`.

[17] H. Durrant-Whyte and T. Bailey, „Simultaneous localization and mapping: part I,“ *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, Jun. 2006.

[18] F. Alhwarin, A. Ferrein, and I. Scholl, „IR stereo kinect: improving depth images by combining structured light with IR stereo,“ in *Pacific Rim International Conference on Artificial Intelligence*, Springer, 2014, pp. 409–421.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, „You Only Look Once: Unified, Real-Time Object Detection,“ in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.

[20] B. Scholz, „Object Reconstruction,“ 2017.

[21] „An overview of deep-learning based object-detection algorithms,“ in. [Online]. Available: `https://medium.com/@fractaldle/brief-overview-on-object-detection-algorithms-ec516929be93`.

[22] Z. Zhao, P. Zheng, S. Xu, and X. Wu, „Object Detection With Deep Learning: A Review,“ *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2019.

[23] D. Pangercic, V. Haltakov, and M. Beetz, *Fast and Robust Object Detection in Household Environments Using Vocabulary Trees with SIFT Descriptors*,

[24] D. G. Lowe, „Distinctive image features from scale-invariant keypoints,“ *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[25] Yan Ke and R. Sukthankar, „PCA-SIFT: a more distinctive representation for local image descriptors,“ in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2, Jun. 2004, pp. II–II.

[26] N. Dalal and B. Triggs, „Histograms of oriented gradients for human detection,“ 2005.

[27] D. G. Lowe *et al.*, „Object recognition from local scale-invariant features.,“ in *iccv*, vol. 99, 1999, pp. 1150–1157.

[28] W. Zhao and C. Ngo, „Flip-Invariant SIFT for Copy and Object Detection,“ *IEEE Transactions on Image Processing*, vol. 22, no. 3, pp. 980–991, Mar. 2013.

[29] Y. LeCun, Y. Bengio, and G. Hinton, „Deep learning,“ *Nature*, vol. 521, 436â€“444, May 2015. [Online]. Available: `https://doi.org/10.1038/nature14539`.

[30] R. Girshick, J. Donahue, T. Darrell, and J. Malik, „Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation,“ in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[31] R. Girshick, „Fast R-CNN,“ in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[32] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, „Fast 3d recognition and pose using the viewpoint feature histogram,“ in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010, pp. 2155–2162.

[33] W. Wohlkinger and M. Vincze, „Shape distributions on voxel surfaces for 3d object classification from depth images,“ in *2011 IEEE international conference on signal and image processing applications (ICSIPA)*, IEEE, 2011, pp. 115–120.

[34] A. Aldoma, F. Tombari, R. B. Rusu, and M. Vincze, „OUR-CVFH–oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6DOF pose estimation,“ in *Joint DAGM (German Association for Pattern Recognition) and OAGM Symposium*, Springer, 2012, pp. 113–122.

[35] W. Wohlkinger and M. Vincze, „Ensemble of shape functions for 3d object classification,“ in *2011 IEEE international conference on robotics and biomimetics*, IEEE, 2011, pp. 2987–2992.

[36] X.-F. Hana, J. S. Jin, J. Xie, M.-J. Wang, and W. Jiang, „A comprehensive review of 3d point cloud descriptors," *arXiv preprint arXiv:1802.02297*, 2018.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, „Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[38] S. Ren, K. He, R. B. Girshick, and J. Sun, „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *CoRR*, vol. abs/1506.01497, 2015. arXiv: `1506.01497`. [Online]. Available: `http://arxiv.org/abs/1506.01497`.

[39] K. He, G. Gkioxari, P. Dollár, and R. Girshick, „Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[40] J. Dai, Y. Li, K. He, and J. Sun, *R-FCN: Object Detection via Region-based Fully Convolutional Networks*, 2016. arXiv: `1605.06409 [cs.CV]`.

[41] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, „Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[42] „Comparison of object detection algorithms," in. [Online]. Available: `https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359`.

[43] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, „Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[44] G. Gemignani, R. Capobianco, E. Bastianelli, D. Bloisi, L. Iocchi, and D. Nardi, „Living with robots: Interactive environmental knowledge acquisition," *Robotics and Autonomous Systems*, vol. 78, Jan. 2016.

[45] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, „OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[46] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger, „SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks," *CoRR*, vol. abs/1609.05130, 2016. arXiv: `1609.05130`. [Online]. Available: `http://arxiv.org/abs/1609.05130`.

[47] J. McCormac, R. Clark, M. Bloesch, A. J. Davison, and S. Leutenegger, „Fusion++: Volumetric Object-Level SLAM,“ *CoRR*, vol. abs/1808.08378, 2018. arXiv: `1808.08378`. [Online]. Available: `http://arxiv.org/abs/1808.08378`.

[48] M. Tenorth and M. Beetz, „KnowRobâ€”knowledge processing for autonomous personal robots,“ in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009, pp. 4261–4266.

[49] M. Beetz, F. BÃ¡lint-BenczÃ©di, N. Blodow, D. Nyga, T. Wiedemeyer, and Z. MÃ¡rton, „RoboSherlock: Unstructured information processing for robot perception,“ in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1549–1556.

[50] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, „Towards 3D Point cloud based object maps for household environments,“ *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008, Semantic Knowledge in Robotics, ISSN: 0921-8890. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0921889008001140`.

[51] „Robot operating System,“ in. [Online]. Available: `https://www.toyota-global.com/innovation/partner_robot/robot/`.

[52] „Robot operating System,“ in. [Online]. Available: `ros.org`.

[53] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, „ROS: an open-source Robot Operating System,“ in *ICRA Workshop on Open Source Software*, 2009.

[54] „ROS Tutorials from Clearpath Robotics,“ in. [Online]. Available: `https://www.clearpathrobotics.com/assets/guides/ros/`.

[55] A. S. Kundu, O. Mazumder, A. Dhar, and S. Bhaumik, „Occupancy grid map generation using 360°scanning xtion pro live for indoor mobile robot navigation,“ in *2016 IEEE First International Conference on Control, Measurement and Instrumentation (CMI)*, Jan. 2016, pp. 464–468.

[56] D. M. Swoboda, „A comprehensive characterization of the asus xtion pro depth sensor,“ in *I: European Conference on Educational Robotics*, 2014, p. 3.

[57] R. B. Rusu and S. Cousins, „3D is here: Point Cloud Library (PCL),“ in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1–4.

[58] R. Szeliski, „Algorithms and Applications,“ *Computer Vision, Springer-Verlag New York, Inc*, pp. 201–13, 2010.

[59] „ROS Tutorials from Clearpath Robotics," in, Dec. 2018. [Online]. Available: `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[60] J. Redmon and A. Farhadi, „YOLO9000: Better, Faster, Stronger," *CoRR*, vol. abs/1612.08242, 2016. arXiv: `1612.08242`. [Online]. Available: `http://arxiv.org/abs/1612.08242`.

[61] ——, „Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[62] „Object Detection YOLO v1 , v2, v3," in, Jan. 2019. [Online]. Available: `https://medium.com/@venkatakrishna.jonnalagadda/object-detection-yolo-v1-v2-v3-c3d5eca2312a`.

[63] J. Redmon, *Darknet: Open Source Neural Networks in C*, `http://pjreddie.com/darknet/`, 2016.

[64] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, „ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255.

[65] „darknet yolo," in. [Online]. Available: `https://pjreddie.com/darknet/yolo/`.

[66] M. Simon, S. Milz, K. Amende, and H.-M. Gross, „Complex-YOLO: An Euler-Region-Proposal for Real-Time 3D Object Detection on Point Clouds," in *Computer Vision – ECCV 2018 Workshops*, L. Leal-Taixé and S. Roth, Eds., Cham: Springer International Publishing, 2019, pp. 197–209, ISBN: 978-3-030-11009-3.

[67] „Symbol for mongoDB," in. [Online]. Available: `https://github.com/FortAwesome/Font-Awesome/issues/13197`.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Datum:

Unterschrift:

Name: