An Autonomous Mobile Handling Robot Using Object Recognition

Johannes Nikolaus Rauer, BSc. – 1710331019

***Abstract*—Due to the trend away from mass production to highly customized goods, there is a great demand for versatile robots in the manufacturing industry. Classic fixed-programmed industrial robots and rail-bound transport vehicles that can only transport standardized boxes do not offer enough flexibility for modern factories. New machine learning methods and 3D vision sensors can give manipulators the ability to perceive and understand their environment and therefore perform tasks such as part recognition or pose estimation. Giving a manipulator furthermore the ability to drive on flexible paths in a production facility, such a robot can be used to detect, grasp and transport individual objects autonomously.**

This report describes the development of an automatic transport robot using a sensitive manipulator and 3D vision sensors for the autonomous transport of objects. This mobile manipulator is able to localize and classify predefined objects in an industrial environment, perform path and movement planning, drive to the object and grasp and transport it. Furthermore, it provides a platform to test new algorithms and procedures for grasp detection and calculation.

***Index Terms*—Deep Learning, Object Recognition, Mobile Manipulator**

I. Introduction

SINCE the beginning of the 1960s, traditional fixed-base robots have been used in factories [1]. These stationary manipulators have mainly been deployed in mass production, where a constant environment and physical setup can be assumed and thus highly engineered programs can work efficiently. The adaptation of these control-programs to new conditions is difficult and expensive [2], [3].

Due to the lack of sensing capabilities, classic industrial robots are unable to deal with the uncertainties in the real world – for example if a part is not located exactly at the position that the robot assumes. Therefore, tasks which could not be modeled sufficiently could not be performed with robotic manipulators in history [4]. Giving manipulators the ability to perceive the environment and using these perceptions for machine learning, flexible part recognition and pose estimation in unstructured environments is possible. Training of these algorithms on new products gives such manipulators the flexibility to react to new conditions quickly and cost-effectively [5].

Due to the trend away from mass production to highly customized goods, there is a great demand for versatile robots in the manufacturing industry [6], [7]. Over the last few years e.g. the production of cars has been highly individualized. As Pavlichenko et al. describe, this has made "kitting" necessary, where all parts of a car are collected in a warehouse and brought to the assembly line just in time as a "kit". This task is frequently performed by warehouseman due to the high flexibility needed to find, collect and transport specific components [8]. Therefore, part handling during assembly stages is the only task in the automotive industry with an automation level below 30% [6].

For such intra-logistical transport tasks, mobile manipulators have been developed. They consist of sensitive manipulators which are associated with mobile transport vehicles to combine the advantages of both types of robots: Working together with humans and grasping individual objects, as well as changing the location autonomously to extend the workspace of the manipulator [7], [9]. A mobile manipulator should be able to drive autonomously through a shop floor, detect parts with sensors, grasp them with its manipulator and transport the goods to production facilities. It thereby leads to a higher level of automation which increases productivity and quality [8].

This paper presents the development of a mobile manipulator using object recognition to grasp and transport objects in an industrial environment. A mobile platform is therefore extended with a sensitive manipulator, a gripper and an image sensor. The developed mobile manipulator is able to plan paths and move to defined goals, search for objects using an image stream and machine learning, move the manipulator to the detected object, as well as grasp and transport it. Furthermore, it provides a platform to test new algorithms and procedures for grasp detection and calculation.

The following chapters are structured as follows: Section II presents the state of the art of 3D-vision-systems, grasp detection and machine learning with a focus on neural networks. In section III, the software representation of the robot, the calibration of the camera and the implemented software are described, as well as the hardware and structure of the mobile manipulator. Section IV explains the abilities of the robot and verifies and discusses them using experiments, followed by the summary and future work in section V.

II. State of the Art

The problem of grasping consists of perceiving objects, recognizing graspable regions and calculating their real world positions [10], [11]. Therefore, in the following section, the relevant state of the art for these tasks is discussed.

A. 3D Vision

The first step to enable grasp detection is perceiving the robot's environment. Especially visual sensors play an important role in object detection and manipulation. 3D vision systems, which map the 2D image pixels to 3D world coordinates,

are most suitable if they are mounted to the arm of a robot [12], [13].

There are three main 3D vision systems on the market which are suitable for such an application: Time-of-Flight (TOF), structured-light and active stereoscopy, whereby the last two use a triangulation process to estimate the depth. TOF-systems measure depth-distance directly from the time a light beam needs until it is reflected back [14]. Active stereoscopy devices have the advantage over classical stereoscopic depth systems, that they project an infrared pattern and are therefore able to find matching points for the triangulation also on texture-less surfaces [15]. 3D cameras often include an RGB-sensor and therefore provide RGB-D images [14].

B. Machine Learning

Using these images and machine learning, graspable objects can be recognized. To extract information such as object types and positions from big datasets, neuronal networks are used. They are able to classify data into different categories or extract useful information such as if a specific object occurs in an image [16], [17]. Deep neuronal networks consist of numerous layers with artificial neurons in them, which allows them to handle big data effectively. A convolutional neural network (CNN) is a special type of deep network, which is designed to work with data that has a grid-like topology, such as images [17], [18].

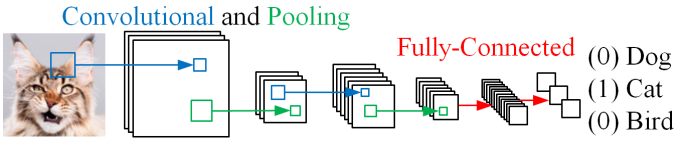

Fig. 1. Typical structure of a CNN: Blue - Convolutional Layer, Green - Pooling Layer, Red - Fully-Connected Layer

A typical CNN, which is presented in figure 1, consists of a series of convolutional and pooling layers which act as feature extractors and fully connected layers for classification. A convolutional layer generates feature maps by doing a convolution between a sliding filter and the input. The dimension of the feature maps is reduced using a pooling layer. The thereby extracted high level features are used for classification using fully connected layers. These produce an n-dimensional output where each describes the probability of the input being of a class [17], [18].

To predict besides class probabilities also bounding boxes for objects in a single-shot, Redmon et al. have presented the CNN YOLOnet[1] (You only look once)[19]. Classic approaches work with a sliding window where a classifier is run over the complete image [17], [18]. More recent R-CNN methods first propose regions where objects may be and then run a classifier on these bounding boxes [20], [21] but due to the high number of individual components, these networks are slower and hard to optimize [19]. YOLO has the advantages that it is extremely fast. YOLOv3 can process images at 45 frames per second on a Titan X GPU [22]. A robot using such a network is enabled to react rapidly to changes in the environment which is particularly important for object manipulation and grasp detection [23].

[1]YOLOnet pre-trained on ImageNet: www.pjreddie.com/darknet/yolo

C. Grasp Detection

The goal of robotic grasp detection is to detect and calculate graspable regions in images of objects and compute a trajectory to them [24], [10], [11]. Due to challenging variations of the objects and light conditions, as well as occlusions and clutter, grasp-detection systems lag far behind human performance levels [25].

Some grasp-detection methods focus on the object pose estimation and lookup of the corresponding grasp points or gripper pose in a database [10], [26], [11]. At other approaches, neural networks are trained to provide the full grasp configuration [23], [27], [28] or the success-probability for a grasp in a given gripper pose [29], [30]. Due to the creation of an appropriate training dataset being currently a big obstacle, numerous approaches generate data automatically using robots or simulations [29], [26], [28]. Labeling data manually is a challenging and time-consuming task due to objects being graspable in multiple ways and the necessary label being of a high dimension and gripper-dependent [29]. For grasping objects e.g. with a parallel-jaw gripper, a 3D grasp-point is not sufficient. Jiang et al. proposed a 7D representation which also includes a 3D grasp-orientation and the gripper opening width [31]. It has later been simplified to a 5D representation by Lenz et al. – 2D grasp point, rotation, length and width of a grasp-rectangle – assuming that a 2D pose can be transformed into a 3D pose by a robot with an RGB-D camera [24].

Deep CNNs are usually not designed to work with such data with four channels and there is also very little RGB-D data available for training. One way to deal with such images in neural networks for grasp-detection is to replace the blue channel with the depth data after pre-training with RGB images [23] or to work with an RGB and an RGD network and merge the results [32]. Another approach is to refine the results of RGB-grasp-detection with the help of depth-data to increase accuracy [10].

III. Methods and Implementation

In this section, the representation of the mobile manipulator and its environment with coordinate frames is discussed and the structure of the robot as well as all components are introduced. Furthermore, the implementation of the robot's software is explained and a novel approach for the hand-eye-calibration of the camera is presented.

A. Robot Representation

To be able to manipulate an object, knowing its exact location depending on the robot's and the camera's position and orientation is essential. The presented mobile manipulator consist of a MiR100 mobile platform and an UR5 articulated robot, as well as a gripper and a depth camera (see figure 8). All components are represented using coordinate frames which are connected with each other and with the world-frame via geometrical relations, as could be seen in figure 2.

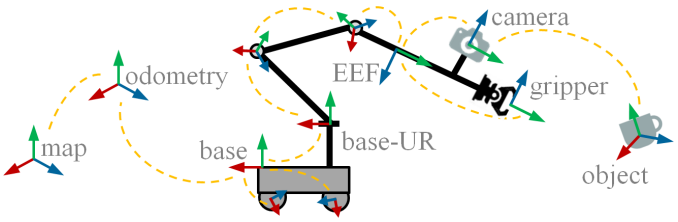

Fig. 2. Positions and connections of the robot's frames. Orientations are just exemplary. To reduce complexity, unimportant frames – e.g. of sensors of the MiR or of the UR's arm-joints – are not shown or labeled. The dashed lines illustrate the mutual dependencies of the frames.

The map-frame represents the robot's world coordinate system and functions thereby as fixed origin. As the robot moves, the odometry-frame is moved according to the wheel-odometry and signals from the inertial measurement unit continuously in the map-frame. The base-frame is rigidly attached to the mobile platform and moved in the map-frame according to sensor observations, such as the localization in the map using laser-scanner signals [33]. The base-UR-frame is attached to the base-frame statically and serves as reference for the frames in the joints of the manipulator.

At the end of the kinematic chain, a frame for the tool center point (TCP) of the gripper – in which planning of arm trajectories takes place – as well as a frame for the camera – in which the pose of the object is calculated – are connected statically to the EEF-frame (end-effector-frame).

If an object is presented to the robot, its xyz-position relatively to the camera is calculated. With the knowledge of the transformations between all the other frames, the pose of the object in the robot's world is computable. The transformations between the object-frame and the EEF-frame are determined by calibrating the camera.

B. Calibration of the Camera

To calculate the pose of the object relatively to the robot, two main transformations have to be known: The transformations between the EEF-frame and the camera-frame (extrinsic), as well as between the camera-frame and the object-frame (intrinsic).

Intrinsic parameters are necessary to calculate 3D metric information from 2D images. Cameras are usually calibrated as Zhang (1999) proposed, by showing a planar pattern to a camera in at least two different, unknown orientations [34]. Such a calibration is not necessary if the camera is calibrated by the manufacturer [15].

Determining extrinsic parameters is also known as hand-eye calibration, which aims to find the transformation between the robot's gripper (hand) and the camera (eye). It is necessary due to the sensor in the camera's case may be slightly tilted and the camera itself may not be mounted to the arm exactly right-angled [35].

Classic approaches for extrinsic calibration formulated the hand-eye calibration problem as solving the homogeneous transform equation $AX = XB$, where A is the known change of the wrist position, B is the accompanying displacement of the sensor and X is the sensor's position relative to the wrist. By moving the robotic arm into two or more known positions, a system of equations which is solvable for X results [35], [36]. There exist also newer and more accurate yet more complex solutions for extrinsic calibration without a calibration target [37], [38].

A new approach has been developed which enables a fast and simple calibration, if the translations between the gripper and the camera can be measured precisely in the CAD data and therefore only the rotations have to be determined. Inaccuracies of the translations have only linear effects – as e.g. 1 mm offset of the camera corresponds to a shift of 1 mm in a distance of 2 m – whereas angular rotations lead to significant bigger errors – e.g. 1° incline of the camera leads to an error of 35 mm in a distance of 2 m.

The approach is based on the assumption, that if the sensor in a camera-housing is tilted, the recorded image section is shifted. This shift is measurable relatively to the axis of the camera-housing and therefore the angular error can be calculated using trigonometric functions, as could be seen in figure 3.

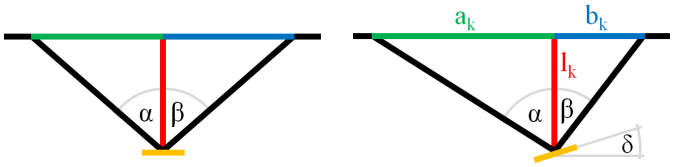

Fig. 3. Left: Camera chip (orange) oriented parallel to housing: $\alpha = \beta$. Right: Camera chip tilted in housing: $\alpha \neq \beta$. With the measured distances a_k, b_k and l_k, the angular error of the sensor δ can be calculated

It is assumed, that the camera is not rotated around the principal axis. The rotations around the other two axes are calculated separately with the same scheme: The robot is moved as near as possible to a planar surface with the flange being oriented exactly parallel to it. The distance between the surface and the gripper is measured and the position of the gripper jaws is marked on the surface. Then the image of the camera is viewed and the borders of the visible image are marked as well on the surface. Thereby, a relationship between the gripper – which has a known position relatively to the camera-housing – and the image-borders is created. A calculation with these parameters would be possible but to increase accuracy, the robot is moved orthogonally away from the surface and the image borders are marked in different distances to enable averaging.

Knowing the position of the camera-housing relatively to the gripper, the positions of the image borders relatively to the camera are calculable. Using the shifts of the image borders from the center of the camera a_k and b_k and the distance of the camera from the surface l_k, the view-angle of the chip to the left and to the right α and β can be calculated with the resulting triangles (see figure 3 right). The accuracy of the results rises, if the mean of all angles in different distances from the surface is calculated, as imprecisions balance each other:

$$\bar{\alpha} = \frac{1}{n}\sum_{k=1}^{n} \arctan\frac{a_k}{l_k} \quad \text{and} \quad \bar{\beta} = \frac{1}{n}\sum_{k=1}^{n} \arctan\frac{b_k}{l_k} \qquad (1)$$

The total view-angle would be calculated by adding up the two angles. Therefore, the incline of the sensor δ can be calculated as followed:

$$\delta = max(\bar{\alpha}, \bar{\beta}) - \frac{\bar{\alpha} + \bar{\beta}}{2} \tag{2}$$

For example if $\bar{\alpha}$ is 25° and $\bar{\beta}$ is 27°, the angular error δ would be $27 - \frac{25+27}{2} = 1°$.

The same procedure has to be done for the rotation around the other axis respectively. With these angles and the translation measured in CAD data, the transformation between the robot's gripper and the camera can be formulated.

When performing the calibration with this method it has to be ensured, that the image borders are marked very accurate as they directly affect the result of the calibration. Therefore, the distance between the camera and the surface should not get too big as marking the borders exactly gets more difficult.

C. Implementation of the Software

If the camera is calibrated exactly, the robot's representation is complete and can be used to grasp objects using appropriate software. The software of the mobile manipulator is based on ROS (Robot operating system) [39], where an industrial computer serves as main ROS-core, which handles the messages of all components. Upon startup of the mobile manipulator there are different nodes launched to provide basic communicate with the hardware and to the core. The structure of these basic nodes is presented in figure 4.

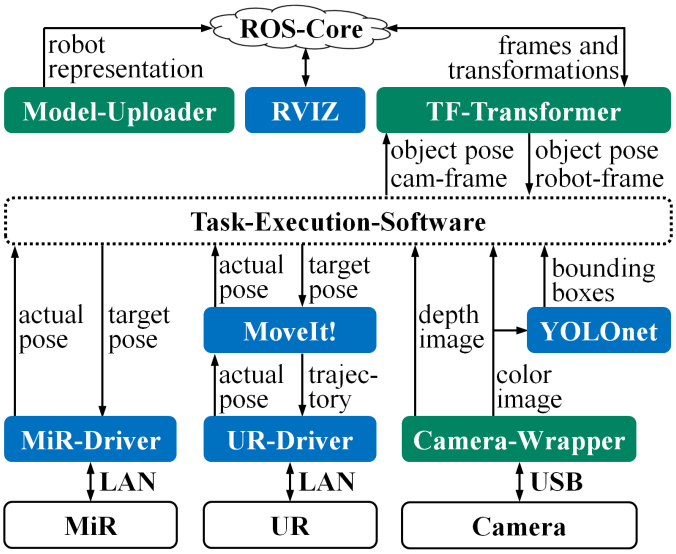

Fig. 4. Structure of the basic nodes of the software, which provide topics to communicate with the hardware. Green nodes have been developed. Blue nodes symbolize provided ROS-Packages.

When the task-execution is started, the connections to the basic nodes are established as illustrated. The program flow of the task-execution is presented in figure 5 and explained in section III-C2. The basic nodes are described in the following section:

1) Basic Nodes:

The basic nodes are responsible for providing communication with the hardware and elementary information of the robot, such as its kinematic structure. Therefore, a robot model including its structure and the coordinate frames is sent to the core by the Model-Uploader. The ROS-core provides it to all nodes to enable visualizations, the representation of the coordinate frames in the tf-tree[2] and the calculation of trajectories. Both robots and the gripper are modeled as URDF-Files[3] which are connected with XACRO[4] and the result is uploaded to the parameter server as "robot_description".

Using this robot-description, tf[2] builds a tree as illustrated in figure 2. The module TF-Transformer writes transformations to tf and calculates important transformations and publishes them as ROS-topics. It broadcasts to tf static transformations which are not modeled in the URDF-file – e.g. the camera-position relatively to the robot's wrist. Since tf calculates transformations at specific times, broadcasting e.g. the objects position during normal program flow at random moments could lead to problems. Therefore, the main program publishes the position to a ROS-topic and TF-Transformer calculates and broadcasts the transformation to tf at a constant rate. Also transformations necessary for planning – e.g. the object-pose relatively to the UR base – are calculated periodically and published as ROS-topics.

To compute these transformations, the UR's arm angles are necessary. Therefore, a TCP-IP connection to the UR5 is established using UR-Driver[5]. This node provides the actual robot pose and is able to send trajectories to the robot [40]. The ROS-package MoveIt![6] communicates with the UR-Driver and serves as motion planner and kinematics solver for the UR-Robot. It offers a programming interface[7] to get the robots joint states and arm pose. Given a target pose it solves the inverse kinematics and calculates a trajectory[8].

The communication between the industrial PC and the MiR's controller is provided by the ROS-package MiR-Driver[9]. Since the MiR launches its own ROS-core, this bridge is necessary to communicate. MiR-Driver establishes a connection to the MiR, reads all topics and provides the same topics in the local ROS-environment on the industrial computer. It reads all messages of the MiR and publishes them to the local ROS-core – e.g. the actual pose. If something is sent to a MiR-topic – e.g. a goal pose – it is forwarded to the MiR and published to its ROS-core.

In addition to driving and moving the robotic arm, perceiving the environment is necessary to enable the mobile manipulator to grasp objects. Therefore, a Camera-Wrapper connects to the RGB-D-camera and publishes color and depth images

[2] wiki.ros.org/tf
[3] wiki.ros.org/urdf
[4] wiki.ros.org/xacro
[5] www.github.com/ros-industrial/ur_modern_driver
[6] www.github.com/ros-planning/moveit
[7] docs.ros.org/kinetic/api/moveit_commander/html/classmoveit__commander_1_1move__group_1_1MoveGroupCommander.html
[8] moveit.ros.org/documentation/concepts/
[9] www.github.com/dfki-ric/mir_robot

using the camera's Python-API[10]. The CNN YOLOnet[11] subscribes to the color-image-stream and publishes the object-types and -positions in the image frames as bounding boxes [22]. YOLOnet is pre-trained using the ImageNet dataset[1][41]. To visualize the robot, its motion and the environment, the ROS-package RVIZ[12] is implemented. Using the map provided by the MiR and the arm angles of the UR as well as the tf-tree, the robot can be virtually presented to users in its environment.

With these basic services and communications, the actual task execution of driving, searching, locating and grasping the object is possible, which is explained in the following section:

2) Task execution:

The program which is necessary for executing the task itself is started by user-input and calls different modules with special purposes. This makes it possible to change modules and implement e.g. a different grasping algorithm with no effect to other program parts. In figure 5, the general procedure for grasping an object is presented abstractly and following the parts are described in detail.

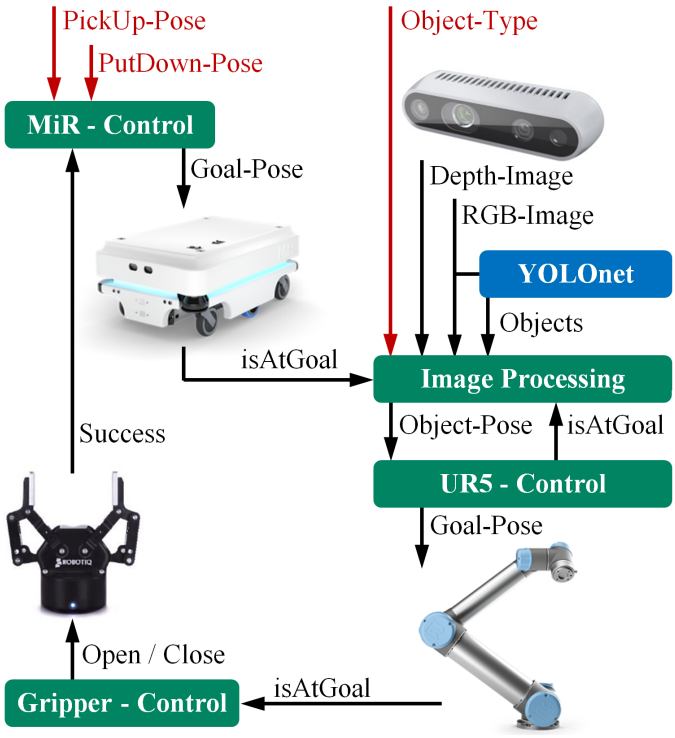

Fig. 5. Program flow of the robot's task-execution-software. Controls all components to grasp an object of given type at given pick-up-position and to transport it to given put-down-position. User inputs are marked red. Green nodes have been developed. Blue nodes symbolize provided ROS-Packages.

A complete task-execution consist of driving to a goal position, searching for the object and calculating the grasp point, as well as moving the robotic arm and grasping the object at the calculated point. Then the robot can move to a goal position and put the object down. For sending the robot to a target pose, the module MiR-Control builds a ROS-Message from a given PickUp-Pose, which is forwarded to the robot. The MiR calculates the path and drives to the goal without further intervention, as it is able to perform tasks such as localization, path planning and obstacle detection autonomously [42]. A second method implements the verification if the robot has reached the prescribed goal using the robot's pose.

When the MiR is at the goal, the pose of the object has to be determined. The camera's RGB-image is analyzed by YOLOnet[11], which offers the object-types and positions of the bounding boxes for each recognized object in the image [19]. Together with the depth-image and the camera intrinsic parameters, the image-processing-module calculates the coordinates of the demanded object in 3D-coordinates relatively to the camera and publishes this information. As the pose estimation is not precisely when analyzing a single shot, the UR is moved multiple times and different image processing steps are performed, which is for reasons of simplification not shown in figure 5. First the UR is moved to different search positions where the camera has a good overview of the scene. If the object is detected, the robot moves about 35 cm over the item with the camera parallel to the plane under the object for grasp detection.

Currently, grasp detection is performed for each object individually and the processes are designed manually. Therefore, the functions are located in the image processing module. To detect the grasping point, the object is separated from the background of the depth-image and circle detection is performed, as could be seen in figure 6. A bottle can be grasped using the located circle's center. The grasp point of a cup is calculated using the center of it and the end of the handle, which is the furthest point from the center.

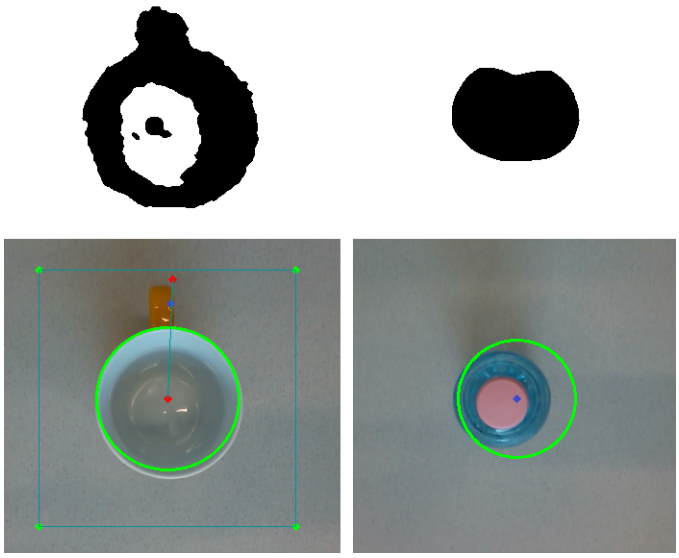

Fig. 6. Detection of the grasp point (blue). Top: Depth-images separated from background. Left: Grasp-Point located at the handle of a cup. Right: Grasp-Point at the center of a bottle.

These grasp points are provided to the tf-tree to calculate the grasp point relatively to the robot's frame. Using this transformation, UR5-Control calculates the goal pose of the UR5 to grasp the object.

[10]www.github.com/IntelRealSense/librealsense/tree/master/wrappers
[11]www.github.com/leggedrobotics/darknet_ros
[12]wiki.ros.org/rviz

After the UR has reached the aimed grasping pose, gripper-control sends a command to the robot to close the gripper. The gripper is connected to the UR to make it controllable with the robot's teach pendant. Therefore, commands have to be sent to the gripper via the UR, as illustrated in figure 7.

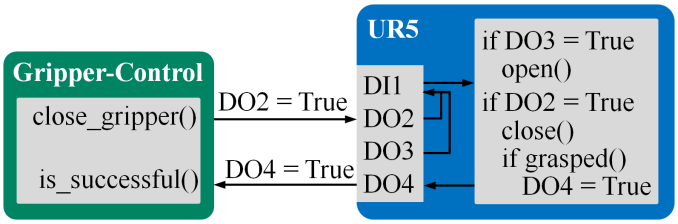

Fig. 7. Communication with the gripper. Gripper-Control sets the robot's digital outputs (DO) via LAN to call a program which opens and closes the gripper. Successful grasps are indicated by another DO signal.

The robot's digital outputs can be set using the LAN-connection to the robot. Whether the gripper is to be opened or closed, a different output is set. Both are connected to an input of the robot which triggers the execution of a program on the robot's control unit. This checks which output is set, opens or closes the gripper respectively and upon successfully grasping the object, it sets another output of the robot. This is read by the gripper control on the industrial computer, so it can be checked if a grasping attempt has been successful. If the object has been grasped by the robot, the MiR can be sent to the PutDown-Pose where the UR5 can lie down the object.

D. Components & Structure of the Robot

In this section, the hardware of the robot is introduced. All components of the robot – which are visible in figure 8 – as well as their connections to each other are shortly described.

The mobile platform which serves as base for all components is a MiR100[13]. It is able to calculate and travel the path to a given goal autonomously and is thereby aware of safety issues. On top of an aluminum structure, a collaborative articulated UR5 robot[14] is installed. This manipulator weights about 20 kg at a payload of 5 kg and a maximum grasp-distance of 850 mm. It is connected to the MiR's Ethernet-switch via LAN.

A 3D-printed fixture with an Intel Realsense D435[15] RGB-D-camera is mounted onto the flange of the manipulator. This active stereoscopy based camera provides RGB-D-images between 0.2 and 4.5 meters in depth at up to 90 frames per second. It is connected with an USB-C cable to the industrial computer, where the image frames can be read using software interface. The parallel-jaw gripper 2F-85[16] is mounted onto the camera fixture at the end of the kinematic chain of the robot. With its payload of 5 kg at an opening width of 85 mm it is suitable to grasp small objects. It is connected to the control unit of the UR5 via USB.

[13] www.mobile-industrial-robots.com/de/products/mir100
[14] www.universal-robots.com/de/produkte/ur5-roboter/
[15] click.intel.com/intelr-realsensetm-depth-camera-d435.html
[16] www.robotiq.com/products/2f85-140-adaptive-robot-gripper

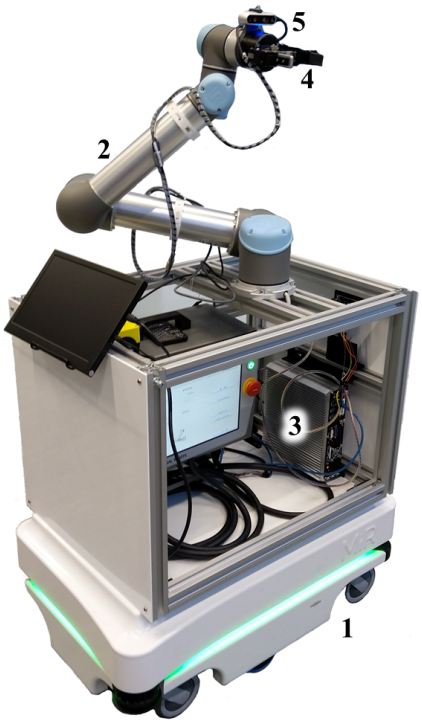

Fig. 8. Structure of the robot: 1 - Mobile Robot. 2 - Handling Robot. 3 - Control Unit. 4 - Gripper. 5 - RGB-D-Camera

To control all subcomponents of the mobile manipulator, the industrial computer ECS-9100-GTX1050T[17] is installed on the mobile robot. With its NVIDIA GeForce GTX 1050 Ti graphics processor it is ideal for image processing tasks. The industrial computer runs on Ubuntu 16.04 and hosts a ROS-Kinetic-core[18]. It is connected to the Ethernet-switch of the MiR and thereby able to access and control all components.

IV. Results & Discussion

In the following section, the abilities of the robot are described and verified using experiments, focusing on the quality of the calibration of the camera and the object detection. Furthermore, reasons for problems of the object recognition are presented and discussed.

A. Abilities of the Robot

The presented mobile manipulator is able to localize and classify predefined objects, to perform a path and movement planning, to drive to the object, and to grasp and transport it as well as put it down. The current supported objects are cups and bowls, whereby a software platform has been created which can be extended to manipulate other objects. If a new object is to be added, it is necessary to train the neural network with images of it and to write a method for calculating the grasp point using a depth image. The developed software can due to its modular structure also be used as base for implementing and testing different grasping approaches.

[17] www.vecow.com/dispUploadBox/PJ-VECOW/Files/3238.pdf
[18] wiki.ros.org/kinetic

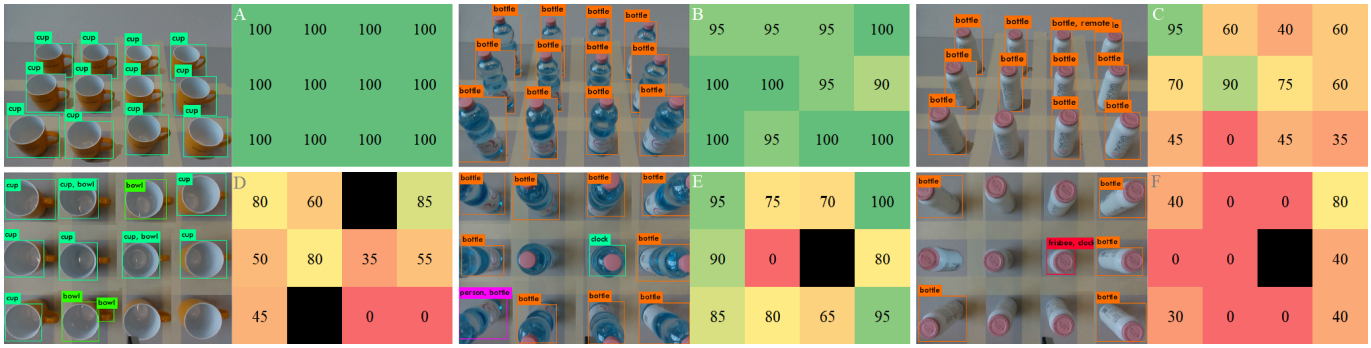

Fig. 9. Quality of object detection: Cup (left), Standard water bottle (center), Unconventional smoothie bottle (right). Camera has been oriented 40° (top) and 90° (bottom). Images show the merged pictures of all positions. Colored fields illustrate the calculated probability of the object being of the correct class. Black fields show wrong detections.

B. Camera Calibration

When grasping an object, calculating the grasp point in the robot's frame exactly is necessary, using parameters of the calibration of the camera. To evaluate the quality of this calibration, an experiment has been performed to determine the accuracy of the calculated calibration angles. The robot grasps the same object multiple times at the same position and under constant light conditions, whereby the orientation of the object is changed at each attempt. This experiment has therefore been performed using a cup. It shows that the robot is able to grasp the object each time at 30 repetitions. The calibration has shown that the image sensor is tilted 1.28° around the vertical axis and 0.58° around the horizontal axis of the camera. When calculating the grasping point, the camera is located about 400 mm over the object. This leads to a linear error of the grasping point without calibration of about 9 mm in the horizontal direction and 4 mm vertically, using trigonometric functions. These errors would make a successful grasp impossible and show the importance of an accurate camera calibration, even if rotations are small.

C. Object Recognition

An object can only be grasped if it is recognized correctly first. The quality of the object detection, depending on the angle of the camera perceiving it and the position of the object in the captured image has been evaluated practically using experiments. For this, the robot has been located in front of a table showing a 4x3 grid under constant lighting conditions. The camera has been oriented by manipulating the robot's arm to show the whole grid at a specific angle. Afterwards, an object has been presented to the camera at the different positions of the grid under constant orientation. It has been evaluated if the object is recognized in the captured image and the quality of this detection has been analyzed using the class-probabilities provided by the neural network. This procedure has been performed at camera angles of 40° and 90° at a distance of 750 and 600 mm to the grid's center respectively, using three different objects: A cup with great contrast, a standard water bottle and an unconventional smoothie bottle. The objects have been oriented with the cup's handle pointing 90° to the right and the bottle's labels pointing to the robot.

The results of these experiments are presented in figure 9. The color of the fields indicate the probability of an object being of the correct class, whereby red marks positions where no class could be determined and black illustrates detections of a wrong class. The detection rate of cups and standard water bottles from a side view are very high, as figure 9 (A, B) shows. Unconventional bottles as captured in (C) are less likely to be recognized correctly. The experiments show significant lower success rates of the object recognition for top view images. Cups are detected less accurate at all locations, whereby especially the lowest positions cause problems. Detecting cups incorrectly as bowls is also a common mistake (D). The detection rate of water bottles from the top view (E) is also smaller than from the side, whereby no correct detection of bottles shown from exactly above at the center positions are observable. This is visible for the unconventional bottles too (F). It is also interesting to note that bottles in the top view images (E, F) at a specific location have always been identified as clocks with a probability of 50%.

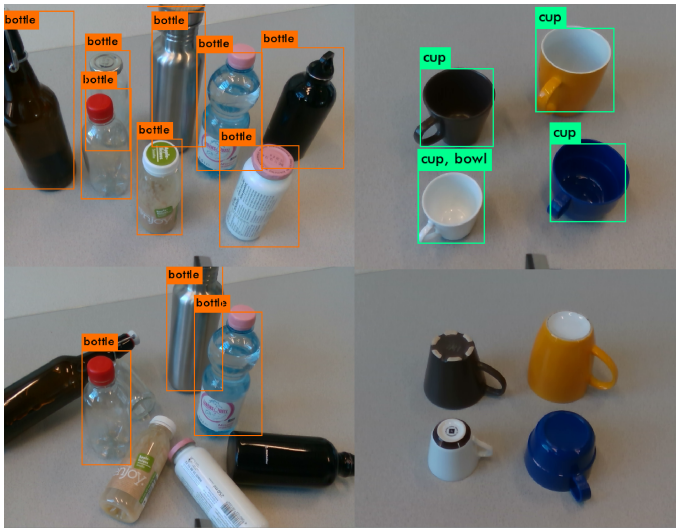

Fig. 10. Different objects captured from various perspectives. The low number of correct detections of objects in unconventional orientations points out weaknesses of the object recognition.

These experiments show clear weaknesses of the used neural network for a robotic application. The ImageNet dataset which

has been used to train YOLOnet[1] is using images found on the Internet [41]. Therefore most of the objects are captured in their natural position as humans see them – e.g. most of the images of cups show medium-sized ceramic mugs with the handle clearly visible from an angle of about 45°. There are only few images showing cups from above, from a flat angle or upside-down[19]. The neural network learns from this insufficient dataset and thereby struggles to generalize to objects in new or unusual poses or configurations. This is also obvious in figure 10, where differently arranged objects are detected with varying quality. Bottles standing on a table are recognized correctly in contrast to the same bottles lying down. Cups are not detected if they are put upside-down.

The results demonstrate the importance of an accurate and application specific training-dataset for neural networks. To enable high detection rates, images of concrete objects to be recognized from real-life perspectives have to be included in sufficient number.

V. Summary and Future Work

In this paper, an automatic transport robot using a sensitive manipulator and 3D vision sensors for the autonomous transport of objects has been presented. This mobile manipulator is able to localize and classify predefined objects in an industrial environment using machine learning, perform a path and movement planning, drive to the object and grasp and transport it. Furthermore, it provides a platform to test new algorithms and procedures for grasp detection and object recognition.

A new approach for the hand-eye calibration of a wrist-mounted camera, using the shift of the image borders to calculate the sensor's orientation, has been introduced and validated with experiments. Besides the presentation of the hardware components and the structure of this mobile manipulator, modular software based on ROS has been described, which has the advantage of the basic nodes and hardware communication being decoupled from the actual task execution software. Experiments have shown that the main weakness is detecting objects reliable in different poses and from varying points of view due to insufficient training of the neural network. This limits the mobile manipulator as object recognition is necessary for grasping objects.

Therefore, in future work the neural network should be trained using a different dataset which shows objects as the robot's camera captures them. The network and grasp detection should furthermore be extended to enable the robot to recognize and grasp a larger variety of objects. It should be considered generating training images using the robot autonomously or synthetic data from simulations. Furthermore, a three-dimensional picture of the robot's environment could be created using its depth camera to increase safety and thereby enable the robot to perform tasks with higher velocity.

[19] www.image-net.org/synset?wnid=n03147509

References

[1] J. Wallén, "The history of the industrial robot," *Technical report from Automatic Control at Linköpings universitet*, 2008.

[2] M. A. Roa, D. Berenson, and W. Huang, "Mobile manipulation: Toward smart manufacturing [tc spotlight]," *IEEE Robotics Automation Magazine*, vol. 22, no. 4, pp. 14–15, Dec. 2015.

[3] J. Sturm, *Approaches to Probabilistic Model Learning for Mobile Manipulation Robots*, ser. Springer Tracts in Advanced Robotics (STAR), B. Siciliano and O. Khatib, Eds. Springer, 2013.

[4] V. Krueger, A. Chazoule, M. Crosby, A. Lasnier, M. R. Pedersen, F. Rovida, L. Nalpantidis, R. Petrick, C. Toscano, and G. Veiga, "A vertical and cyber–physical integration of cognitive robots in manufacturing," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1114–1127, May 2016.

[5] C. H. Corbato, M. Bharatheesha, J. Van Egmond, J. Ju, and M. Wisse, "Integrating different levels of automation: Lessons from winning the amazon robotics challenge 2016," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4916–4926, Nov. 2018.

[6] D. Holz and S. Behnke, "Fast edge-based detection and localization of transport boxes and pallets in rgb-d images for mobile robot bin picking," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, Jun. 2016, pp. 1–8.

[7] H. Hirsch-Kreinsen, "Digitization of industrial work: development paths and prospects," *Journal for Labour Market Research*, vol. 49, no. 1, pp. 1–14, Jul. 2016.

[8] D. Pavlichenko, G. M. García, S. Koo, and S. Behnke, "Kittingbot: A mobile manipulation robot for collaborative kitting in automotive logistics," in *Intelligent Autonomous Systems 15 - Proceedings of the 15th International Conference IAS-15, Baden-Baden, Germany, June 11-15, 2018*, 2018, pp. 849–864.

[9] Z.-E. Chebab, J.-C. Fauroux, N. Bouton, Y. Mezouar, and L. Sabourin, "Autonomous collaborative mobile manipulators : State of the art," in *TrC-IFToMM Symposium on Theory of Machines and Mechanisms*, Izmir, Turkey, 2015.

[10] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes," *Robotics: Science and Systems (RSS)*, 2018.

[11] G. Billings and M. Johnson-Roberson, "Silhonet: An rgb method for 3d object pose estimation and grasp planning," *CoRR*, vol. abs/1809.06893, Sep. 2018.

[12] S. Xie, E. Haemmerle, Y. Cheng, and P. Gamage, "Vision-guided robot control for 3d object recognition and manipulation," in *Robot Manipulators*. InTech, Sep. 2008, pp. 521–546.

[13] M. M. Ali, H. Liu, R. Stoll, and K. Thurow, "Arm grasping for mobile robot transportation using Kinect sensor and kinematic analysis," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, May 2015, pp. 516–521.

[14] S. Giancola, M. Valenti, and R. Sala, "A survey on 3d cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies," in *SpringerBriefs in Computer Science*, ser. SpringerBriefs in Computer Science. Cham: Springer International Publishing, 2018.

[15] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," *Computer Vision and Pattern Recognition*, 2017.

[16] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[17] Q. Abbas, M. E. Ibrahim, and M. A. Jaffar, "A comprehensive review of recent advances on deep vision systems," May 2018.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2016, pp. 779–788.

[20] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1440–1448.

[21] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *NIPS*, 2016.

[22] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[23] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1316–1322.

[24] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, Jan. 2015.

[25] U. Asif, M. Bennamoun, and F. Sohel, "Real-time pose estimation of rigid objects using rgb-d imagery," in *Proceedings of the 2013 IEEE 8th Conference on Industrial Electronics and Applications, ICIEA 2013*. IEEE, Jun. 2013, pp. 1692–1699.

[26] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *IEEE International Conference on Intelligent Robots and Systems*, Vancouver, Canada, Sep. 2017, pp. 545–551.

[27] F.-J. Chu, R. Xu, and P. A. Vela, "Real-world multi-object, multi-grasp detection," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3355–3362, Oct. 2018.

[28] U. Viereck, A. ten Pas, K. Saenko, and R. Platt, "Learning a visuomotor controller for real world robotic grasping using simulated depth images," *CoRR*, vol. abs/1706.04652, 2017.

[29] L. Pinto and A. Gupta, "Supersizing self-supervision : Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016, pp. 3406–3413.

[30] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," in *Robotics: Science and Systems*, Jul. 2017.

[31] Y. Jiang, S. Moseson, and A. Saxena, "Efficient grasping from rgbd images: Learning using a new rectangle representation," in *Proceedings - IEEE International Conference on Robotics and Automation*, May 2011, pp. 3304–3311.

[32] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2017, pp. 769–776.

[33] W. Meeussen, "Coordinate frames for mobile platforms," 2010. [Online]. Available: http://www.ros.org/reps/rep-0105.html

[34] Zhang, Zhengyou, "Flexible camera calibration by viewing a plane from unknown orientations," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 1, 1999, pp. 666–673.

[35] R. Y. Tsai and R. K. Lenz, "A new technique for fully autonomous and efficient 3d robotics hand/eye calibration," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 3, pp. 345–358, Jun. 1989.

[36] Y. C. Shiu and S. Ahmad, "Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form ax = xb," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 16–29, 1989.

[37] X. Zhi and S. Schwertfeger, "Simultaneous hand-eye calibration and reconstruction," in *IEEE International Conference on Intelligent Robots and Systems*, Mar. 2017, pp. 1470–1477.

[38] W. Li, M. Dong, N. Lu, X. Lou, and P. Sun, "Simultaneous robot–world and hand–eye calibration without a calibration object," *Sensors*, vol. 18, no. 11, Nov. 2018.

[39] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *International Conference on Robotics and Automation (ICRA), workshop on open source software*, 2009.

[40] T. T. Andersen, *Optimizing the Universal Robots ROS driver*. Technical University of Denmark, Department of Electrical Engineering, 2015.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[42] Mobile Industrial Robots, *MiR100 User Guide*, Mobile Industrial Robots, 2019.