An Environment-Aware Robot Arm Platform Using Low-Cost Sensors and Deep Learning

Justinas Mišeikis

May 14, 2019

Thesis submitted for the degree of Philosophiæ Doctor

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 2169

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Abstract

The main aim of the thesis is to use low-cost hardware components to create an environment-aware robot arm system capable of analysing its workspace using vision sensors and ensure collision-free operation. The objective is not only to build a physical system, but the focus is to develop algorithms to understand the environment by using image and depth information from multiple cameras and allow the robot to calculate and execute safe movement trajectories in dynamically changing environment.

In this thesis, we have developed an automatic camera-to-robot calibration system to perform camera internal and Eye-to-Hand calibrations. Furthermore, a visually based reactive-reflexive robot behaviour method allows the robot to find safe trajectories throughout a dynamically changing environment with a capability of quickly reacting to unexpectedly appearing close obstacles. The robot was also used to charge electric vehicles by using vision-based guidance autonomously. Eventually, the work evolved to using deep learning approaches to recognise the robot and estimate its 3D position with a simple 2D colour image used as an input. Multi-objective convolutional neural networks and transfer learning techniques allowed to expand the method to more robot types when having a limited amount of training data.

The thesis concludes that commercially-available hardware can be integrated using advanced algorithms to precisely model the workspace of the robot and allow path planning and quick reactions to unexpected situations. Many 3D cameras as well as robots, including Universal Robots, Kuka iiwa LBR and Franka Emika Panda, were used to perform all the experiments in real-world conditions.

Preface

This thesis is submitted to the University of Oslo for the degree of Doctor of Philosophy. The research was conducted at the Department of Informatics, University of Oslo (UiO) and the Intervention Centre, Oslo University Hospital (OUS). The primary supervisor was Professor Jim Torresen. The co-supervisors were Associate Professor Kyrre Glette (UiO) and Associate Professor Ole Jakob Elle (OUS/UiO).

This thesis has been written entirely using Overleaf Online Latex Editor with the majority of the figures created using Keynote. Some of the graphs were generated using Python and Seaborn library.

Acknowledgments

I would like to express my gratitude to all my advisors for the provided support and guidance throughout my PhD study: Jim Torresen, Kyrre Glette and Ole Jakob Elle for helping me out from start to finish, endless discussions regarding the direction of the project, reviewing many iterations of my papers, providing the required hardware and challenging my ideas to make the quality of work even higher.

I am very thankful for an opportunity to spend two semesters at the Robot Vision Group at the Technical University of Graz (TU Graz) in Austria. Under the supervision of Dr. Matthias Ruther and Prof. Horst Bischof, numerous interesting projects were undertaken resulting in successful publications.

I would like to thank my research colleagues Dr. Saeed Yahyanejad, Inka Brijacak and Univ.-Doz. DI Dr. Michael Hofbaur at Joanneum Research: Robotics for their close collaboration, sharing many ideas and an invitation to use their robots for my research. Also, a big thank you goes to Bernhard Walzel at Institute of Automotive Engineering, TU Graz for an efficient collaboration on a robotic electric vehicle charging project.

Most of all, I would like to thank my fiance Nelija Borisenko and my family for supporting through this challenging and rewarding PhD journey.

List of Thesis Papers

Paper I **Automatic calibration of a robot manipulator and multi 3D camera system**

J. Miseikis, K. Glette, O. J. Elle and J. Torresen

2016 IEEE/SICE International Symposium on System Integration (SII), Sapporo, 2016, pp. 735-741.

Paper II **Multi 3D camera mapping for predictive and reflexive robot manipulator trajectory estimation**

J. Miseikis, K. Glette, O. J. Elle and J. Torresen

2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, 2016, pp. 1-8.

Paper III **3D Vision Guided Robotic Charging Station for Electric and Plug-in Hybrid Vehicles**

J. Miseikis, M. Rüther, B. Walzel, M. Hirz and H. Brunner

OAGM/AAPR & ARW Joint Workshop 2017, Wien, Austria.

Nominated for the Best Student Paper Award

Paper IV **Robot Localisation and 3D Position Estimation Using a Free-Moving Camera and Cascaded Convolutional Neural Networks**

J. Miseikis, P. Knobelreiter, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen

2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Auckland, New Zealand, 2018, pp. 181-187.

Finalist of the Best Student Paper Award

Paper V **Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images**

J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle and J. Torresen.

2018 15th International Conference on Ubiquitous Robots (UR), Honolulu, HI, USA, 2018, pp. 597-603.

Paper VI **Transfer Learning for Unseen Robot Detection and Joint Estimation on a Multi-Objective Convolutional Neural Network**

J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen

2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR), Shenyang, 2018, pp. 337-342.

Winner of the Best Student Paper Award

Paper VII **Two-Stage Transfer Learning for Heterogeneous Robot Detection and 3D Joint Position Estimation in a 2D Camera Image using CNN**

J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen

2019 IEEE International Conference on Robotics and Automation (ICRA), Montreal, 2019

Contents

Abstract iii
Preface v
Acknowledgments vii
List of Thesis Papers ix

1 Introduction **1**
1.1 Related Work 2
1.2 Research Objectives 4
1.3 Achievements 5
1.4 Thesis Outline 5

2 Background and Equipment Used **7**
2.1 Hardware 7
2.1.1 Robot Manipulators 7
2.1.2 Sensors: Vision and Motion Capture System 8
2.1.3 Computers 10
2.2 Software 10
2.2.1 Robot Operating System 10
2.3 Robot Control and Path Planning 11
2.4 Camera Calibration 12
2.5 Workspace Sensing and Mapping 14
2.5.1 Point Clouds 14
2.5.2 Octomaps 15
2.6 Deep Learning 16
2.6.1 Convolutional Neural Networks 17
2.6.2 Multi-Objective Convolutional Neural Networks 18
2.6.3 Transfer Learning 19

3 Research Summary **21**
3.1 Overview 21
3.2 Papers 23
3.2.1 Paper I 23
3.2.2 Paper II 26
3.2.3 Paper III 30
3.2.4 Paper IV 34
3.2.5 Paper V 36
3.2.6 Paper VI 38
3.2.7 Paper VII 40

4 Discussion **43**
4.1 Research Approaches and Achievements 43
4.1.1 Collision-Free Robot Operation in a Shared Workspace . . 43
4.1.2 Robot Body and Pose Estimation Using Deep Learning . . 44
4.2 Limitations 47
4.2.1 Robot Limitations 47
4.2.2 Vision Sensor Limitations 47
4.2.3 Deep Learning Limitations 49
4.3 Conclusions 49
4.4 Future Work 51

5 Papers **53**

Bibliography **108**

Chapter 1

Introduction

The first significant impact in changing the manufacturing industry took place in the 1980s during the early industrial robotic revolution [1]. Then, the robots were developed which were stronger and more precise than before. They could work tirelessly without any fatigue symptoms, being especially useful in repetitive movements. Factories were remodelled to incorporate these robots in the production lines allowing to reduce the number of workers needed, reduce manufacturing errors, handle more substantial and heavier parts, and most importantly - reduce the costs.

However, there is an issue regarding the flexibility of industrial robots. They are outstanding in performing pre-programmed movements, but they usually operate blindly and have to be fenced off from the people because they will not notice anything on their way and might cause a severe injury. The same issue is present when the conditions differ even slightly from what is expected and programmed. The robot will not adjust to changing conditions. Lastly, for any adjustment to the setup or the operation program, a qualified robotics engineer has to be present to implement these changes.

However, we are approaching a new robotic revolution, often referred to as Industry 4.0 [2]. It is a result of the hardware costs going down and making both the robotic arms as well as vision sensors more affordable, together with the increasing processing power of computers [3]. It allowed for the development of a new kind of industrial robots: collaborative robots, or so-called cobots [4]. The development and adoption of cobots have seen a steep increase over the last 10 years [5]. The main advantage of cobots compared to traditional industrial robots lies in their safety features. Instead of having a necessity of fencing them off from people, the robots will be to detect any potential impact with a person and

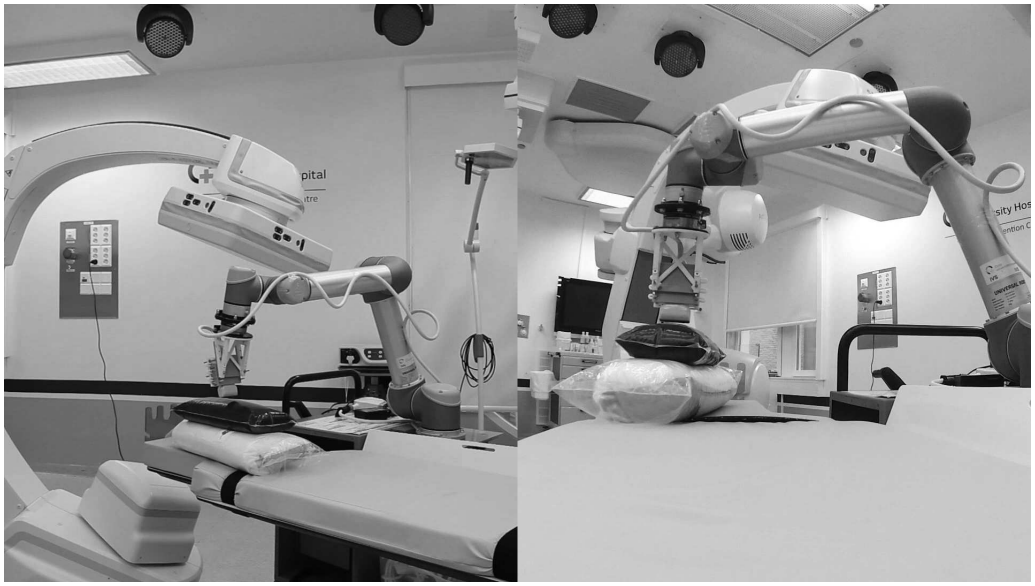


Figure 1.1. UR5 robot holding an ultrasound probe over a dummy patient next to a C-arm mounted flouroscopy scanner. Photo taken in Oslo University Hospital.

perform a safety stop to avoid causing any injury. Furthermore, cobots need a more intuitive programming interface, so hiring a robotics engineer to adjust the behaviour of the robot should not be required anymore.

Despite the significant advances in the robot and sensor development, there is still an outstanding issue of allowing the robot to understand its environment and have a flexible, adaptive behaviour. It is possible to sense the environment of the robot, but often it is difficult to define where the robot itself is, which of the setup components are fixed (like a table-top), and distinguish between the objects of interest and obstacles that should be avoided. Furthermore, the camera-robot setups are often fixed; thus, reconfiguration requires additional work.

1.1 Related Work

Despite the robots being actively used in industry, the shared workspace issue is still being solved despite being researched for many years [6] [7]. Workspace sharing can be classified as *robot-robot* and *human-robot* systems for task sharing, collaborative or supportive tasks. Normally, in *robot-robot* sharing, controllers of all involved robot systems have direct communication and can coordinate moves easier by knowing the planned trajectories for all the manipulators. This thesis

will be focusing on *human-robot* shared workspaces, where sensors are used to observe the environment and adapt the manipulator behaviour according to movements in the workspace, normally caused by human motion.

Many systems have been proposed addressing the trajectory planning in a shared workspace. One example is a system running a genetic algorithm based on fuzzy logic. As all the obstacles are defined as static while the new trajectory is calculated, the method would not be suitable if objects are moving at high velocities [8]. Some of the systems for *human-robot* interaction assign the robot arm as a lightweight manipulator, thus reducing a possible collision force and then using inertia reduction, passivity and parametric path planning [9]. However, this method leads to light collisions, which ideally should be avoided.

Another work has analysed the usage of non-verbal cues given by humans and robots, which would be an equivalent of body language in our daily interactions. It has proven that understanding of movement plays an essential role in the usability of a system and the *human-robot interaction (HRI)* [10].

Cameras are quite commonly used to overlook the workspace of the robot and analyse the environment. Using the visual information, together with the depth data, typically provided by 3D cameras, collision-free trajectories could be calculated. It has been shown that a robot can rapidly react and recalculate the movements when an obstacle blocks the current path [11]. Another system uses multiple Kinect cameras observing the same workspace from different viewing points to avoid collisions, but no particular planning approach was proposed in the paper [12]. Furthermore, a system was introduced using the historical data of obstacle positions in the workspace of the robot. It avoids the areas which are commonly occluded by a human and plans movement trajectories around them [13]. The mentioned works have shown significant progress in the human-robot interaction in the shared workspace. However, the issue remains of having a combination of both reactive and predictive behaviour and ensuring that no contact between the robot and a person will occur.

Another research area that has been increasingly gaining popularity, especially in *natural language processing* (NLP) and *computer vision* (CV), is *deep learning*. It is a field of *artificial intelligence* (AI) based on multi-layer *artificial neural networks* (ANNs) which are capable of self-adjusting the parameters in order to solve a given task [14] [15] [16] [17]. A breakthrough in computer vision was done after the introduction of *ImageNet convolutional neural network* (CNN) revolutionising object classification in images [18]. Many variations and other types of deep learning algorithms were developed since then to solve a variety of tasks.

For example, *recurrent neural networks* (RNNs) are suitable for sequential data like text or speech [19]. They are often based on having *long short term memory* (LSTM), which stores information over extended time intervals by using a recurrent backpropagation method during the training [20].

For computer vision, CNN is typically the best choice, given its ability to analyse an image using spatial filters and choosing the most effective ones for the given task during the training process [21]. However, ordinarily, large amounts of training data have to be used to correctly train the network. Every training sample has to be given *ground truth* (a class label or an accurately estimated value), which is the correct answer for the network. If done manually, an extensive amount of work is needed to generate this data. However, if there is a readily-trained CNN, it can be adjusted for new variants of input data by re-training it using *transfer learning* method [22]. Instead of modifying all the parameters in the CNN, some layers are *locked*, while the rest are adjusted during the training process. It uses the fact that the first layers of the CNN learn more general visual features like edges, corners, gradients and similar so that they can be reused. In the meantime, subsequent layers learn more object-specific features. It has proven to work and requires significantly smaller datasets, and reduces the training time [23].

1.2 Research Objectives

The main aim of this thesis is to take state-of-the-art commercially-available equipment like 3D cameras and collaborative robots and push the boundaries of the environment understanding as well as to allow setups to be quickly reconfigured. All of the work is oriented to allow the developed methods to be used in real-world practical applications.

The work has to be done in a step-by-step manner by proving the general concept first, followed by solving more specific problems in each of the areas. It is a combination of hardware and software solutions, which have to function together to create an environment-aware robotic system.

The main objective of this thesis is defined as:

> Combine *low-cost* sensors and a robot arm to make an *environment-aware robotic system (EAR)* capable of *real-time* operation.

and with the following research questions:

RQ 1. Is it possible to make an *EAR system* to reliably work in a *shared workspace* with a person without a *risk of collision* and injury? More specifically, what precision in obstacle detection and robot movement accuracy can be reached? Would the system be able to react quickly enough to avoid hitting obstacles moving at high velocity?

RQ 2. Are *EAR* systems able to *learn* during the operation from the surrounding environment and *adapt to changing conditions* automatically? To be more specific, can deep learning be applied to this problem and is it possible to train the system using a limited amount of training data?

1.3 Achievements

According to the defined *research questions*, the thesis work resulted in *seven research papers*: six already published at peer-reviewed conferences and one accepted for publication in May 2019.

Research question 1 was addressed by developing an automatic camera-robot calibration system followed by an environment-aware robotic system capable of quickly adapting to changing dynamic conditions in the workspace. It is achieved by having a precise workspace mapping and enabling a predictive-reflexive robot behaviour model. Furthermore, gained knowledge was applied to a project focusing on using a robot arm to detect the charging port and charge electric vehicles automatically.

In the on-going work, the dependency of *Hand-Eye calibration* in camera-robot systems was seen as a limiting factor. It gave inspiration for approaching *research question 2*. The work started by testing *deep learning* approach to identify the robot in a 2D camera image and estimate its position in 3D space. Successful results motivated further research in this area resulting in using a more complex *multi-objective convolutional neural network* and expanding the system to recognise more robots by using *transfer learning* approaches.

1.4 Thesis Outline

This thesis is a collection of papers, and the seven included research papers constitute the research contribution of the thesis. Chapter 2 reviews relevant methods

and approaches that were used throughout the work to develop state-of-the-art techniques as well as the hardware and software used. Chapter 3 gives an overview of the research contributions of this thesis. In Chapter 4, the main findings of this thesis are discussed together with the conclusions. The thesis is finalised with the collection of published papers.

Chapter 2

Background and Equipment Used

In this chapter, the background, as well as the hardware used will be described to provide an overview and motivation of the chosen approaches.

2.1 Hardware

(a) UR3, UR5 and UR10 (b) Kuka iiwa LBR R800 (c) Franka-Emika Panda

Figure 2.1. Robots used in this thesis.

The hardware used in the project can be divided into two major parts: robot manipulators and vision sensors. A brief description is given about each of the hardware components.

2.1.1 Robot Manipulators

Robot manipulator is the centrepiece of the work presented in this thesis. The majority of work has been done with a UR5 robot developed by Universal Robots,

Table 2.1. Specifications of the robots used in this thesis.

Specs	UR3	UR5	UR10	Kuka	Franka Panda
Weight, kg	11	18.4	28.9	23.9	18
Payload, kg	3	5	10	7	3
Reach, mm	500	850	1300	1	855
Joint ranges, ± °	360	360	360	120-175	166
Joint Speed, °/s	180-360	180	120-180	98-180	150-180
Repeatability, mm	±0.1	±0.1	±0.1	±0.1	±0.1
DoF	6	6	6	7	7

given direct access to this robot in the home lab. However, during the exchange period at *TU Graz* and through the partners at *Joanneum Research: Robotics* the following robots were also used: UR3, UR10, Kuka iiwa LBR R800 and Franka-Emika Panda. Each of the robots is of a different size, degrees-of-freedom, operational speed and possible payload, as well as the appearance. All of the robots can be seen in Figure 2.1 and their specifications are summarised in Table 2.1. Diversity in visual appearance and specifications are especially useful for the implementation of deep learning methods to reduce the chance of overfitting the system for one type of robot. Furthermore, having 7 DoF adds redundancy for the robot control; however, it makes the inverse kinematics more complicated.

2.1.2 Sensors: Vision and Motion Capture System

Table 2.2. Technical Specifications of Kinect V1 and V2, Intel Realsense R200 and F200.

	Kinect V1	**Kinect V2**	**Intel R200**	**Intel F200**
Sensor type	Struct Light	ToF	Active Stereo	Coded Light
RGB Cam Res	640x480	1920x1080	1920x1080	1920x1080
IR Cam Res	320x240	512x424	480x360	640x480
Refresh Rate	30 Hz	30 Hz	60 Hz	30 Hz
Depth Range, *m*	0.4 to 4.5	0.5 to 4.5	0.5 to 3.5	0.2 to 1.2
FoV Horizontal	57°	70°	59°	73°
FoV Vertical	43°	60°	46°	59°

For environment sensing, an approach to use vision sensors was taken. Given the reduced cost of 3D cameras, which provide both colour and depth images, a selection of such cameras was used. Additionally, a motion capture system was used to precisely track the position and orientation of the cameras and the robot in

the environment. The system also provided the relative position of tracked objects in relation to each other. It has proven useful when collecting ground truth data for the datasets.

Figure 2.2. Vision sensors used in the projects: 3D cameras and *Optitrack* motion capture system.

The following vision sensors were used: Kinect V1, Kinect V2, Intel Realsense F200 and R200 cameras. Specifications of the vision sensors can be found in Table 2.2. The cameras and *Optitrack* motion capture system is shown in Figure 2.2 [24]. The main differences between the cameras lie in the depth detection method, with *structured light* having interference with several cameras overlooking the same area, while *time-of-flight* (ToF) method projects IR patterns at varying frequencies avoiding the interference problem. The Intel F200 has a smaller depth range making it more suitable for short-range detection. Furthermore, the field-of-view differs between the cameras.

2.1.3 Computers

Given that many of the developed algorithms were optimised for GPU, the main machines used in this thesis contained two *nVidia GTX 1080 Ti* graphics cards and were equipped with *Intel i7* processors and at least 16 GB of RAM. However, many of the algorithms were tested on other machines with *nVidia GTX 1060* and *GTX 1070 Ti* GPUs and were still capable of running in real-time, but at a slightly lower frame rate.

2.2 Software

All the work in this thesis has been running on *Linux*, specifically *Ubuntu* operating system, versions 14.04 and 16.04. Other software packages used in this thesis were: OpenCV, Point Cloud Library (PCL), Robot Operating System (ROS), Ceres Solver, Theano framework with Lasagna, Tensorflow and some prototyping was done in MATLAB. Many open-source algorithms were tested and evaluated from public GitHub repositories, usually published by other researchers.

2.2.1 Robot Operating System

Robot Operating System (ROS) is a meta-operating system running on *Ubuntu*, which provides many algorithms and methods for robotics applications as well as integration with many robotic platforms and sensors [25]. Furthermore, it supports a modular design, which allows to efficiently distribute parts of the algorithm on multiple machines and integrate the whole system to work simultaneously. It is achieved by automatically synchronising all the communication between the modules.

ROS includes a number of libraries useful for environment-aware robotic applications. These include *OpenCV* [26] for computer vision applications, *Point Cloud Library (PCL)* [27] for processing point clouds, *MoveIt!* [28] for robot integration and motion control and *ROS Industrial* [29] aimed specifically at industrial robot arms.

The whole work of this thesis was based on ROS and its subsystems, as it allowed to use many packages out of the box, significantly reducing the time needed to integrate sensors to the robot, develop robot path planning and connect the whole system altogether.

2.3 Robot Control and Path Planning

The control of the robot was done by using official ROS libraries for Kuka and Franka Panda robots, and *UR Modern Driver* [30] for all three of the Universal Robots. It was used to calculate inverse kinematics and execute movements directly on the robot. Movement trajectory calculations were based on *MoveIt!* integration of the *RRT-Connect* algorithm [31], which has proven to be the most efficient planners considering the structure of the robot arm, variety of movements needed and calculation speed.

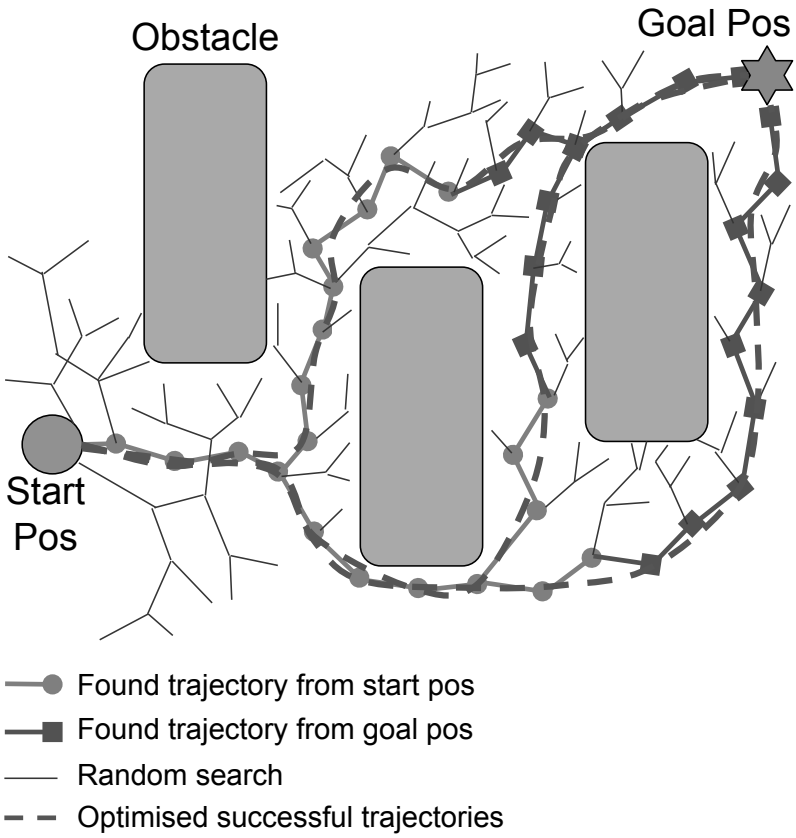


Figure 2.3. Visualisation of a two-dimensional *RRT-connect* trajectory planning algorithm. The method is based on growing Rapidly-exploring Random Trees (marked in thin blue) from the start and goal positions until a connected path is found (visualised in pink and purple). Once one or more successful trajectories are found, they are optimised and smoothed (green dashed lines) before they are executed on the robot.

RRT-Connect motion planner implementation is used for the Cartesian trajectory planning. *RRT-Connect* stands for *Rapidly-exploring Random Trees* (RRTs).

The method works by incrementally building two Rapidly-exploring Random Trees rooted at the start and the goal configurations respectively. Both trees explore space around them independently and also advance towards each other through the use of a greedy heuristics until they connect joining together into a path from the starting point to the goal. A two-dimensional example of *RRT-Connect* algorithm can be seen in Fig. 2.3.

For the trajectory planner to work successfully, the obstacles are precisely modelled in the environment. Then, the free space is defined by calculating all the points, which can be successfully reached given that no part of the robot collides with the obstacles. Then, two RRTs are initialised, both at the start and goal positions and grown in the free space. The exploration uses the randomly-assigned direction and magnitude of vectors, but they are biased towards the goal position as well as unexplored areas. If the tree reaches the goal position or meets the other tree grown from the opposite direction, the successful trajectory has been found. When one or more successful trajectories are found, they are smoothed to avoid choppy robot movements. The most efficient path is executed for the robot to reach the goal position successfully.

2.4 Camera Calibration

Cameras provide beneficial visual information for the computer or robotic system and can be used to detect objects, make measurements or take preventative actions to avoid hitting an obstacle.

To achieve good precision using the information coming from a 3D camera, calibration is typically needed. The calibration for the camera-robot system can be divided into three main categories:

- Internal camera calibration, like lens distortion, focal length, optical center, and for RGB-D cameras, colour and depth image offsets [32] [33]. Typically, a calibration is performed by taking an object of a known structure, like a checkerboard, and it is moved around in front of the camera, while distortions and distance to the points are calculated. It allows creating a calibration matrix, which corrects the differences between the desired position of the object points and the recorded ones.

- External camera calibration: when the sensor comprises of multiple cameras, like two stereo cameras or, in the case of a 3D camera, colour and

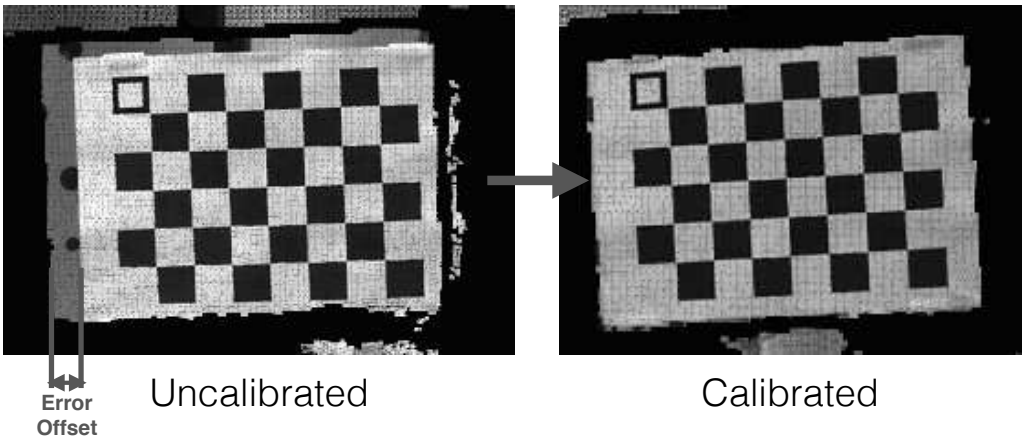


Figure 2.4. Reprojection error is shown in the colour image and depth point cloud overlay. The offset in the left image is caused by imprecisely defined relative positions between the colour and infrared cameras in the 3D camera. Internal camera calibration compensates for this error. The result is seen in the image on the right side, where the offset is reduced.

infrared camera, a relative position between the two cameras has to be precisely calculated. It allows mapping the colour image information on top of the depth or infrared image. Normally, cameras come pre-calibrated; however, the precision is rarely sufficient for applications of high precision. During the calibration procedure, the object like a checkerboard is recorded in both cameras simultaneously, and the transformation between the sets of the same points in two camera images is calculated, and error minimised. The error of between-the-camera calibration is called a *reprojection error*, which has to be minimised. An example of the *reprojection error* before and after the calibration is shown in Figure 2.4.

- Camera to the robot (or object) calibration: the pose (position and orientation) of a camera in a coordinate reference frame. It is commonly called *Hand-Eye* calibration [34] [35]. The *Hand-Eye* calibration or transformation from the camera coordinate system to the robot base coordinate system is shown in Figure 3.2(a). The calibration results in an homogeneous transformation comprised of the rotation component R and translation component t as defined in Equation 2.1.

$$T_{C_3}^{R} = \left\{ \begin{array}{cc} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{array} \right\} \tag{2.1}$$

Compared to the internal camera calibration, usually fewer frames are needed

for *Hand-Eye* calibration and either a similar checkerboard or visually distinct features of a rigid object can be used as keypoints.

2.5 Workspace Sensing and Mapping

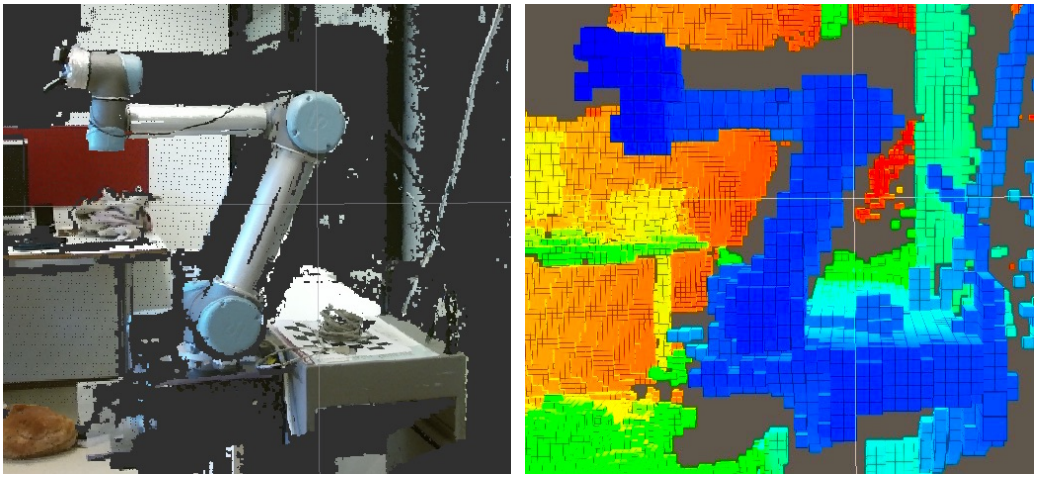
The first step, to allow operations of the environment-aware robot, is to sense the workspace of the robot and create a model of it. It can be done using a variety of sensors, but in this work, we limit ourselves to visual sensing. Different cameras can be used to achieve the goal, including simple colour cameras, stereo cameras and 3D cameras, which provide not only a colour image but also depth image. Depth image indicates the distance from the sensor to various points in the space.

Furthermore, having multiple cameras overlooking the workspace from different angles have proven beneficial to avoid obstacle obstructions as well as to improve the accuracy of the measurements and remove *blind spots*. If the robot itself is seen in the images, it is essential to identify it. Otherwise, the system might classify it as a collision object by estimating a false self-collision. It can be done by using a *self-filtering algorithm*, which takes the robot pose information from the robot motor encoders together with the robot model and filters it out from the camera images [36]. It requires a fully calibrated camera-robot system using *Hand-Eye* calibration.

2.5.1 Point Clouds

The dimensionality of the data is increased by having the depth information from laser scanners, stereo cameras or 3D cameras, compared to a traditional colour camera. On top of the typical X and Y information representing the coordinates, we also have Z for depth, as well as possibly pixel colour information in grayscale or colour. One of the most popular representations of such data is by using point clouds. The recorded points are mapped into the 3D space by using transformations provided by the camera, and they can be visualised efficiently. An example of a point cloud of the UR5 robot can be seen in Figure 2.5(a).

On the other hand, the amount of data increases exponentially with each added dimension, meaning that point cloud data for the equivalent frame is significantly larger compared to a simple colour image information of the same frame. Additional data can become an issue when dealing with many frames from multiple cameras, both regarding the processing speed and storage space requirements.



(a) Point cloud (b) Octomap representing the input point cloud

Figure 2.5. Octomap created from the Kinect V2 point cloud data of the robot scene. colour-scheme represents the distance of objects from the 3D camera.

2.5.2 Octomaps

A more concise volumetric representation is needed to process all the point cloud data provided by multiple cameras in real time. However, the desired precision and resolution should be maintained to allow precise operations. When merging point clouds, it was observed that there are many points which are closely overlapping and adding the density of the points representing the same part of the object, which is not necessarily needed. The representation of the robot workspace using octomap is shown in Figure 2.5(b).

Octomaps are based on an octree hierarchical data structure, which can map the volumetric space by using cubic volumes, defined as voxels [37] [38]. The visualisation of the octree structure is seen in Figure 2.6. The volume is recursively subdivided into eight subvolumes until a given minimum voxel size is reached. This minimum voxel size determines the resolution of the octree. Since an octree is a hierarchical data structure, the tree can be cut at any level to obtain a coarser subdivision.

The main benefit of using an octomap is that only certain parts of the space, like areas containing points of interest, can be represented in high resolution, while coarse resolution can be used for the rest. Furthermore, it automatically combines nearby points from multiple point clouds into the same voxel, which is equivalent to using filtering, resulting in higher processing speed. Having octomap representation of the robot workspace allows processing the complex data

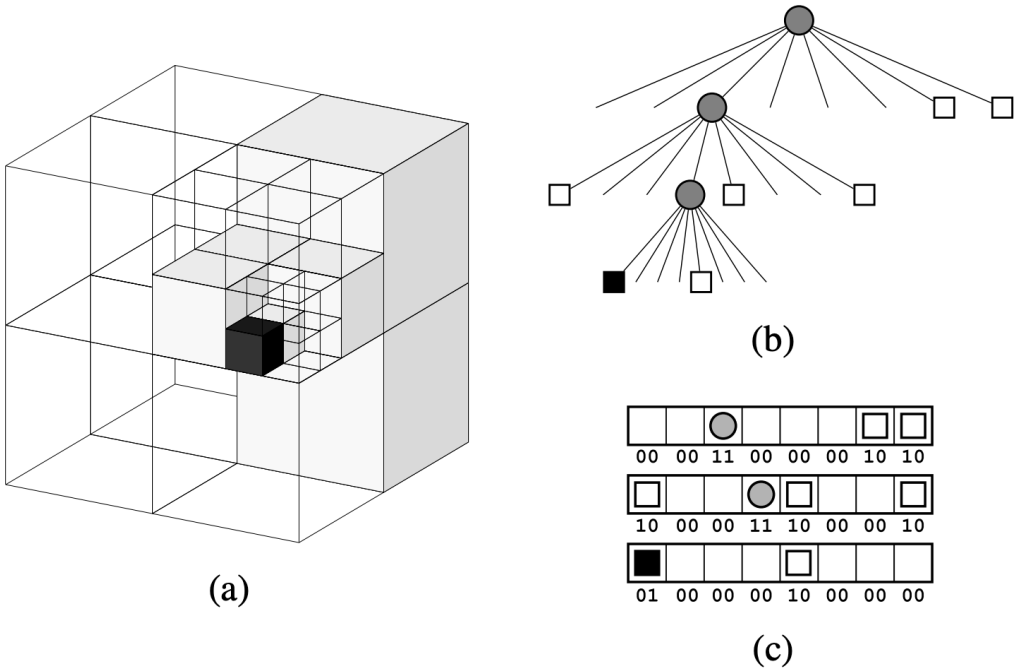


Figure 2.6. a) Octree structure representation. White voxels define empty space, while black voxel represents occupied space. b) The corresponding tree representation. c) The data structure of the corresponding tree is taking up just 6 bytes. Figure is taken from [37]

in a more efficient manner reaching real-time performance.

2.6 Deep Learning

Deep learning is a computational model which allows learning representations of data with increasing levels of abstraction by using multiple processing layers [17]. These machine learning methods have significantly improved visual object detection, speech recognition and generation as well as other domains based on large amounts of data. Deep learning is capable of processing large amounts of data and learn to recognise specific patterns and intricate data structure by using backpropagation algorithm, which indicates to the network which parameters should be changed and how to provide a more accurate answer. Deep convolutional neural networks (CNNs) are typically used to process images, videos, speech and audio. Recurrent nets (RNNs) are used for sequential and time series data analysis and forecastings such as speech and text.

2.6.1 Convolutional Neural Networks

Given that mainly the visual information was being processed in work described in this thesis, convolutional neural networks provided the best performance on the collected data. CNNs are suitable when the data to be learned is provided in a format of arrays, which can have one or multiple dimensions. Images are supplied as 2D arrays if grayscale and 3-layered 2D arrays for colour images.

CNNs typically consist of many interconnected layers placed in sequence, mainly containing convolutional and pooling layers. The network is generally finished with one or two fully connected layers, which eventually connect to output values to provide an answer to the given problem. An example of a CNN structure can be seen in Figure 2.7. However, the structure of CNN can be application-dependent.

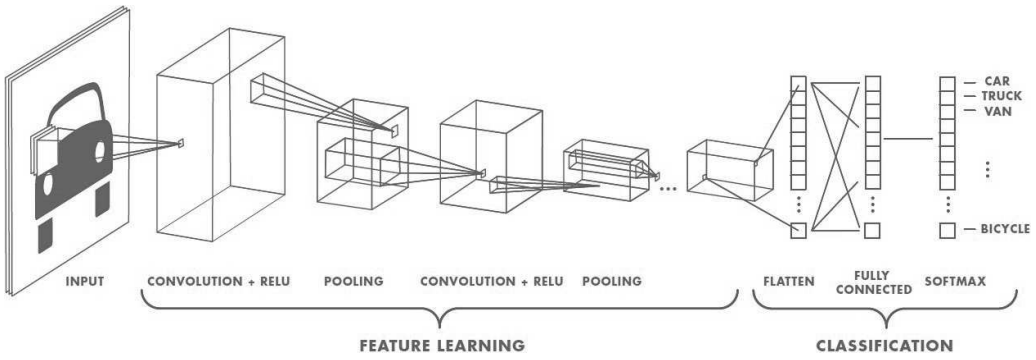


Figure 2.7. Example structure of the Convolutional Neural Network (CNN). Image source [39]

Deep neural networks exploit the idea of compositional hierarchies, where higher level features are composed of lower level features, which is the reason for a sequential structure. Similar approaches can be found in other signal processing applications.

Units in a convolutional layer are organised in feature maps, within which each one is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this locally-weighted sum is then passed through a non-linearity such as a ReLU. The same filter bank is shared among all units in a feature map. Different feature maps in a layer use different filter banks. The reason for this architecture is twofold. First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. Second, the local statistics of images and other signals are invariant to location. In other words, if a motif can

appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array.

Between the convolutional layers, pooling layers can be used. First of all, it reduces the dimensionality of the layers, thus reducing the number of parameters that have to be adjusted. Furthermore, it adds robustness, especially regarding the relative position of the detected features by selecting the most distinct ones in the local area.

The training is done by providing images together with the correct classification labels, correctly marked mask or other information. If the input images are too large to fit into the memory of the GPU, they are sub-divided into mini-batches and fed to the network for training one mini-batch at a time. At each epoch, the result using current network parameters is estimated and an error calculated by using the defined loss function. Then, using this information, the backpropagation algorithm adjusts the parameters of the neural network to reduce the error. The process is repeated until no more improvements are seen or the desired accuracy is achieved.

Loss function allows CNN to evaluate the quality of the learning process by penalising the network for deviations between the predicted output and the actual desired output. The loss is explicitly defined for the type of output and considers possible values, so it is often application-specific.

2.6.2 Multi-Objective Convolutional Neural Networks

Normally the CNN optimises for one type of objective, like classification or regression. However, there are occasions, when having outputs of multiple types would be beneficial. For example, let it be a face recognition task. The primary objective would be to identify who is the person in an input image. On top of that, we can have a face localisation to say where accurately in the picture the face is located, emotion recognition and even gaze direction estimation can provide useful data. To achieve this, CNN has to be able to estimate and optimise for multiple heterogeneous objectives.

To solve this issue, a method called "multi-objective convolutional neural networks" was developed. It uses a similar structure to a standard CNN. However, it provides branching off to multiple outputs. Branching off can be done either at the final fully-connected layer or in the middle of the CNN with some additional objective specific layers before the output layer. An example structure of a

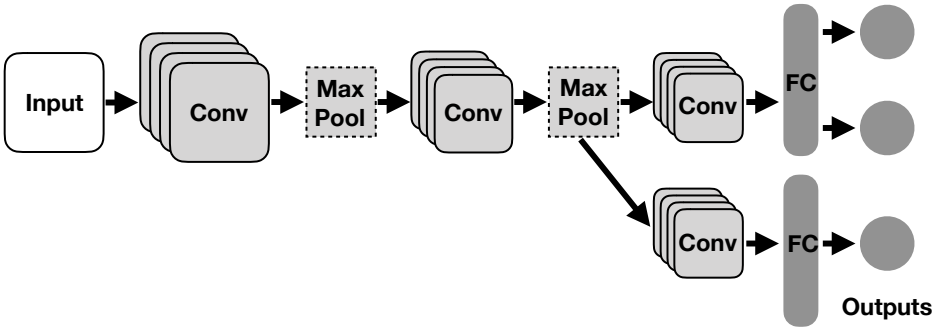


Figure 2.8. Example structure of the Multi-Objective Convolutional Neural Network (CNN).

multi-objective CNN can be seen in Figure 2.8.

The loss function for a multi-objective CNNs has to comprise a combination of losses, one for each of the objectives. One approach is to have a weighted sum of the losses, where weight can represent the importance, or the impact, of each of the outputs to the whole system. CNN is trained simultaneously for all of the objectives.

2.6.3 Transfer Learning

One problem of training CNNs is that large amounts of well-marked and diverse training data are required to successfully train the network. It also results in rather long training processes even on powerful GPUs. It is mainly caused by having large amounts of parameters that have to be tuned at each epoch.

However, when we consider the compositional hierarchy of the CNN and some observations that the first couple of initial layers commonly learn general visual features. These include edges, corners and contrasts of the image. It has been noticed that when given one trained CNN, it is possible to re-train it for a different domain by adjusting just some of the layers of the network instead of all. Typically, some closest layers to the output layer are adjusted. This approach is called *transfer learning* [22] [40].

The benefit of transfer learning technique is that the parameters contained in so-called *frozen* layers are copied from the previously-trained network, while only some of the layers are trained during the process. It speeds up the training process and requires smaller training datasets compared to the full CNN training, while still capable of reaching the equivalent accuracy compared to the fully trained CNNs.

Chapter 3

Research Summary

This chapter will give an overview of the PhD research progress and achievements by the publications.

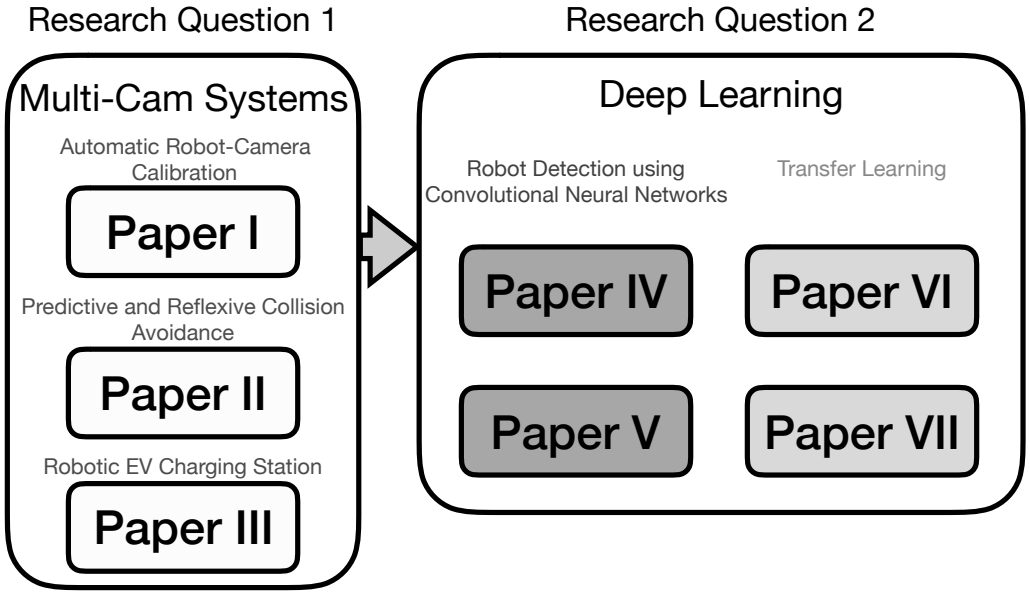


Figure 3.1. Publication overview graph with indications of research questions addressed by each paper or paper group.

3.1 Overview

In total, six papers have been published, with the seventh one accepted to be published in May 2019. Figure 3.1 visualises the topics and research questions ad-

dressed by each publication.

The work started by setting up a robot-camera system for robot workspace mapping and understanding. Such a setup requires a precise *Hand-Eye* calibration to allow the cameras and the robot to function in a *common* coordinate system. A novel method was developed to perform this calibration automatically, resulting in a Paper I described in Section 3.2.1.

Having a precise workspace map recorded as a *point cloud*, it was converted to an octomap to allow for *real-time* obstacle detection. Having this information, obstacle movement trajectories were estimated, and a predictive and reflexive robot manipulator behaviour was developed to quickly react and avoid any static or dynamic obstacles, as published in a Paper II described in Section 3.2.2.

While on an exchange at *TU Graz* in Austria, knowledge from the first two publications was applied to a real project - a robot-based *electric vehicle charging* station. It required a multi-step movement approach from *Paper II*, as well as some automatic calibration features from *Paper I*, but using the charging plug as calibration keypoints instead of a defined marker. This work is described in Section 3.2.3.

Dependency on a *Hand-Eye* calibration used in the first three papers was seen as a possible issue when working with physical systems, which have to be reconfigured. It gave inspiration for *Paper IV*, described in Section 3.2.4, which tested deep learning approaches to detect a robot and estimate its joint coordinates using a simple 2D colour image from the camera, without the *Hand-Eye* calibration.

This work was extended further to use a single multi-objective CNN instead of a cascaded CNN system to achieve the same performance. The system was trained once for three types of Universal Robots at once, instead of having multiple CNNs for each of the robot types. This resulted in *Paper V* described in Section 3.2.5.

To extend this method to more robot types without having a full training cycle, a *transfer learning* approach was proven to work to re-train the system for a Kuka robot arm significantly faster than a complete training process, as described in *Paper VI* in Seciont 3.2.6.

The last publication, *Paper VII* expands the transfer learning method not just to retrain to the new robot type, but by using a two-stage approach, the system can be trained to include new, more complex robots like Franka Emika Panda, still keeping the recognition for previously-trained robots. The system has proven to work for UR, Kuka and Franka Panda robots all at once, as described in Section 3.2.7.

3.2 Papers

This section presents details of the motivations and contributions of each paper.

3.2.1 Paper I

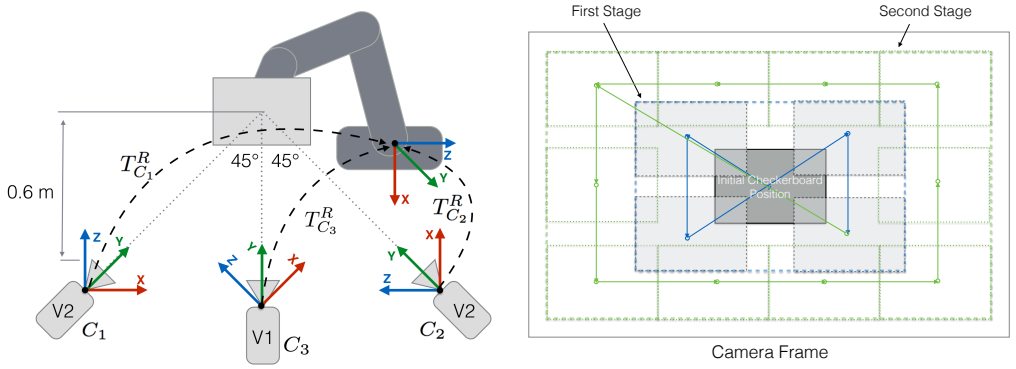
Automatic Calibration of a Robot Manipulator and Multi 3D Camera System

The first step to enable the robot for environment-aware operation is to sense the workspace. Many types of sensors can be used; however, considering the price of the sensors and the information provided by them, 3D cameras, or so-called RGB-D sensors are very suitable in environment-aware robotic applications. On top of the colour image provided by the visual camera part, depth information from the camera to detected objects is also provided. Various representations like point clouds, octomaps, coloured depth maps can be derived by using this information.

To accurately use the camera in the same system with the robot, it has to be calibrated. There are two types of calibration that need to be performed:

- Internal camera parameters, like lens distortion, focal length, optical centre, and for RGB-D cameras, colour and depth image offsets [32] [33].
- External camera parameters: the pose (position and orientation) of a camera in a coordinate reference frame. It is commonly known as *Hand-Eye* calibration [34] [35]. The *Hand-Eye* calibration or transformation from the camera coordinate system to the robot base coordinate system is shown in Figure 3.2(a). This calibration allows for recalculating any object found in the camera's field-of-view, thus coordinate frame, to the coordinate frame of the robot. For example, if the robot needs to grasp or avoid an object detected by the camera.

Usually, an internal calibration needs to be performed only once for each sensor, unless the hardware of the sensor is modified; for example, a lens is changed. On the other hand, the *Hand-Eye* calibration has to be repeated every time the sensors or the robot are moved relative to each other. It is crucial to have this calibration done precisely for the system to have good accuracy in sensing, especially for precise robot operations or visual servoing. Visual servoing uses visual information as a feedback loop to control the robot. It is achieved by calculating the error between the position where the robot end-effector is expected to be in the image compared to the actual location of it in image pixel coordinates.



(a) System Setup with three cameras. In the system, *Hand-Eye* calibration is represented by the Affine transformation matrix T^R_C, which transforms the coordinate system of each camera to the coordinate system of the robot base, making it common for the whole setup.

(b) Robot movement trajectory as seen in the 3D camera image. It is split into multiple stages by the positions calculated at increasing distance from the centre point of the image. Movements are done stage-by-stage while improving the *Hand-Eye* transformation accuracy at each step. This figure shows just a two-stage example.

Figure 3.2. Robot-camera setup and calibration movement trajectory.

Provided that most of the experiments for this thesis were done in a shared lab. A possible issue in the given environment is the risk that the setup can be shifted, accidentally or on purpose, resulting in an imprecise and often invalid *Hand-Eye* calibration. The process of performing this calibration traditionally can easily take 20 to 60 minutes, depending on a number of vision sensors used in the setup. It motivated the idea of creating an automated solution for camera calibration in robot-camera setups.

The calibration marker - a checkerboard was mounted on the UR5 robot using a custom designed, and 3D printed end-effector. The robot was used to move the checkerboard around to calibrate the cameras. The primary constraint for the system to function correctly is to make sure that each of the cameras is overlooking the robot itself together with its workspace. During the initialisation procedure, the robot performs a rotational movement around its base axis, while each of the cameras is trying to detect a checkerboard in its field-of-view (FoV). Once the checkerboard is detected, the robot joint configuration is saved for that position to be later used as a starting point for the calibration movements.

Once the checkerboard has been detected in all the cameras, the calibration procedure can start for each one at a time. First of all, the robot moves to the starting point, where the checkerboard is in front of the camera. Then, the tilting

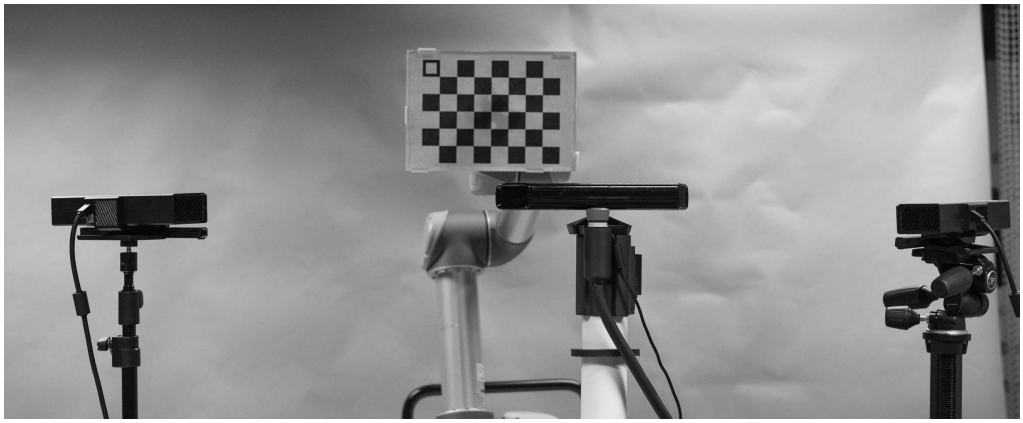
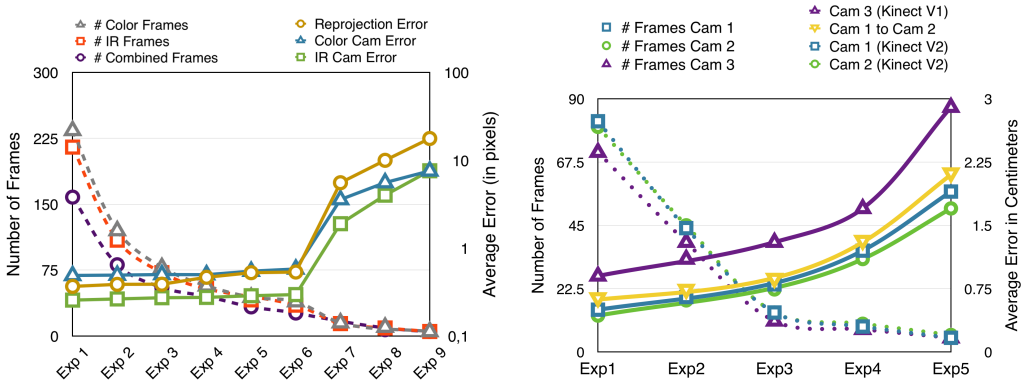


Figure 3.3. Picture of the actual experimental setup. A checkerboard with a hollow square to allow the detection of its orientation is attached to the robot.

motion is performed to record different angles of the board followed by the trajectory of movements with offsets in all of the directions from the starting point. The movement trajectory, as seen in Figure 3.2(b), is guided by the camera, similarly to the visual servoing concept, which allows covering the largest area of the camera's field-of-view. At each position, a new frame is captured by the camera and together with the actual robot position information from the joint encoders is added to the calibration data to improve the accuracy further. Then the next trajectory point is recalculated using the improved calibration and actions repeated.

Many experiments were conducted to test the efficiency of the system. Each experiment used the dataset consisting of a different number of frames to calibrate the camera. They were also divided into two categories, one for the internal camera calibration and another for the *Hand-Eye* calibration. Experiments were conducted by using a three camera setup consisting of two Kinect V2 cameras and one Kinect V1 placed around the robot, as shown in Figure 3.3. Results of the internal camera calibration were evaluated by calculating the reprojection error, and *Hand-Eye* calibration was assessed by calculating an error by the predicted robot position using a camera image versus the actual position based on the robot joint encoder information. The results of the internal camera calibration can be seen in Figure 3.4(a) and the results of the *Hand-Eye* calibration are shown in Figure 3.4(b).

The method has been proven to work, and an optimum number of frames needed for each type of calibration was found. For the internal camera calibration, at least 50 frames for each of the sensors are required to achieve the reprojection error under 1 pixel, and at least 30 frames are needed for accuracy of under 1 cm in



(a) Calibration accuracy results by showing the errors of internal 3D camera calibration. Colour camera, IR errors define averages of each sensor's cameras. Reprojection error determines the average error of the offset in the images between the colour image and the depth information. It has to be noted that right Y axis for error rates is in log scale.

(b) Hand-Eye calibration accuracy results. Overall position error (in cm), as well as each axis separately, are shown by comparing the actual robot position versus the predicted robot position from the 3D camera sensor. Dotted lines indicate the number of frames used in each experiment.

Figure 3.4. Evaluation of the presented automatic calibration process.

Hand-Eye calibration. Calibration time using the automatic method can be under 3 minutes to achieve acceptable accuracy using 50 frames for the internal calibration. For the *Hand-Eye* calibration of the system with three cameras overlooking the robot, the error of under 1 cm was achieved in under 2 minutes of operation for all the cameras. It is approximately ten times faster compared to performing these calibrations manually.

The automatic calibration system was used continuously in the following work when cameras were moved around and had to be re-calibrated.

The work resulted in an automatic calibration system for the robot and multi-camera systems. It allows reducing the calibration time from 20-60 minutes down to 2-10 minutes with close to none of the manual work.

3.2.2 Paper II

Multi 3D Camera Mapping for Predictive and Reflexive Robot Manipulator Trajectory Estimation

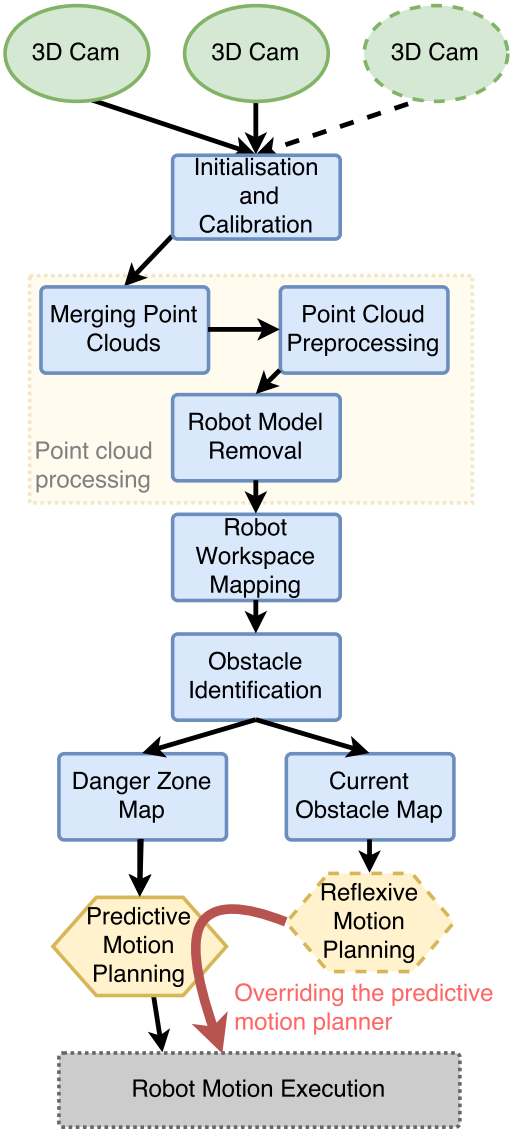
Having a system consisting of a robot manipulator and multiple calibrated 3D cameras allows the system to observe the workspace and detect moving obstacles. By utilising this information, the robot can adapt to changing dynamic conditions.

This work aims at solving the issue of relatively slow trajectory recalculation process of existing methods when an unexpected obstacle appears in very close proximity to the robot, which sometimes can lead to a collision.

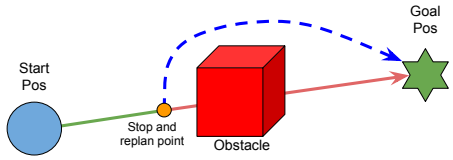
Our setup consisted of having in total four different 3D cameras. It allowed for redundancy as well as ensuring the workspace is seen from many angles, so obstacles do not obstruct the field-of-view. However, with every camera providing a point cloud, the amount of data to be processed was too large for a single computer to provide real-time performance. To address this issue, but still keeping the necessary accuracy, the point clouds were pre-filtered, then merged by using the calibration data, and position fine-tuning using the *iterative closest point* (ICP) method and then converted to an *octomap*. Octomaps can have a different resolution in different parts of the map allowing to have a finer resolution on the points of interest, while background objects, which are out of reach for the robot could be set to have a coarse resolution, thus saving a significant amount of processing power.

Once the map of the workspace was created, the robot itself had to be removed from the octomap to prevent false-positive self-collision instances. It has been done by taking its joint encoder information and fitting a simplified model using finely fit cylinders and overlaying it in the volumetric octomap. Any voxels that are covered by the estimated robot model were removed from the octomap. What was left was a collision map representing static structure obstacles, like a table and a robot cart, as well as any dynamic obstacles that come into a field-of-view of the cameras.

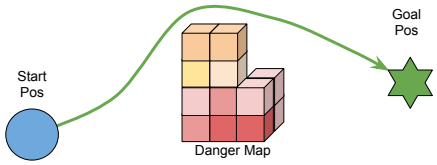
By using a long-term observation of the workspace, a danger map was constructed. It was based on analysing the frequency of the objects occupying the workspace and a cost function based on a logarithmic decay calculates the likelihood that this part of the map will be free or obstructed long-term. By utilising this information, the *predictive behaviour* part of the robot motion planner calculates the trajectory, which takes into consideration the length of the path together with the risk that the area being crossed might be obstructed. The planner is based on the *RRT-Connect* algorithm [31].



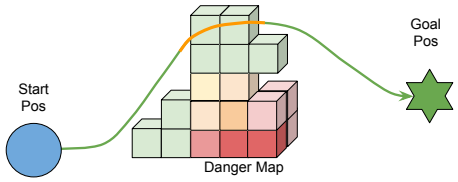
(a) Overview of the algorithm. Green ovals represent the sensing part, blue rectangles - processing part, yellow hexagons - motion planning and grey rectangle (dashed borders) - motion execution. Reflexive motion planning marked as a yellow hexagon (dashed borders) overrides the predictive motion planning when an unexpected obstacle gets close to the robot.



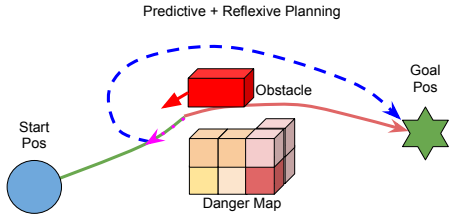
(b) Reactive Behaviour Planning: When the obstacle is present, the robot stops and recalculates the path. Occupied workspace is never crossed.



(c) Our method: the trajectory fully avoids, but gets close to the medium risk area in the danger map. If detour is not large, a safe path is chosen over a risky one.



(d) Our method: the trajectory crosses the low risk area in the danger map.



(e) Our method: Predictive and Reflexive Planning combined, when an unexpected obstacle gets very close to the robot.

Figure 3.5. Algorithm overview for the robot reflexive-predictive behaviour model.

To react quickly and avoid any unexpected obstacles that appear in a very close proximity to the robot, a *reflexive behaviour* model was used. It was inspired by how many of us would pull back an arm if we touch anything hot or sharp, even without understanding or observing what happened. Instead of calculating a new path, the model retraces some previously-executed trajectory points, while a new and safe route is being recalculated using a predictive model in parallel. Before the execution, a check is made to ensure that retracing the previous path is free of obstacles. Otherwise, a movement vector facing away from the detected obstacle is used as a *reflexive behaviour* trajectory. The algorithm is explained in Figure 3.6.

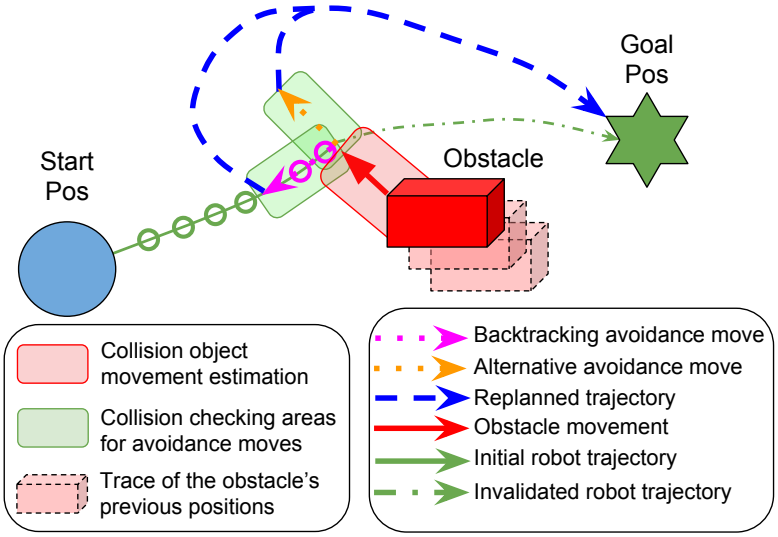


Figure 3.6. Reflexive Behaviour. The planned robot trajectory (solid green line) is blocked by a moving obstacle (marked in red). The first option is to backtrack on the executed path (pink dotted line) until the collision risk is over. If backtracking still results in a collision, the second option is to use the alternative avoidance move and move to the direction opposite from where the obstacle is approaching (dotted orange line). In the meantime, an alternative collision-free trajectory to the goal position is calculated and executed (blue dashed line).

In normal operation, the predictive and reflexive behaviour models are combined. The predictive one is used typically, but in case of an obstacle getting very close to the robot, the reflexive one can override the movement if necessary. The whole process workflow can be seen in Figure 3.5(a).

Many experiments were carried out to evaluate the functionality of the danger map construction, predictive behaviours, reflexive behaviour separately as well as analysing the operation of the whole system. The movements were fixed to be repeated continuously between two points in the pattern A - B - A and with some

static obstacles as well as the dynamic ones occasionally entering the workspace. We compared a simple reactive behaviour, our proposed predictive model, a direct (not-obstructed) path as well as our proposed combined predictive and reflexive behaviour model. It has been shown that our proposed method provides on average three times quicker execution compared to a reactive-only behaviour and around 20% faster compared to our predictive-only behaviour based on a danger map. No collisions were observed by using our proposed method.

The work resulted in a fully working concept, which takes the shortest path considering a calculated risk of possible collisions. In case there is an unexpected obstacle coming close to the robot, a reflexive movement overrides the motion plan, and the robot quickly moves away from the danger.

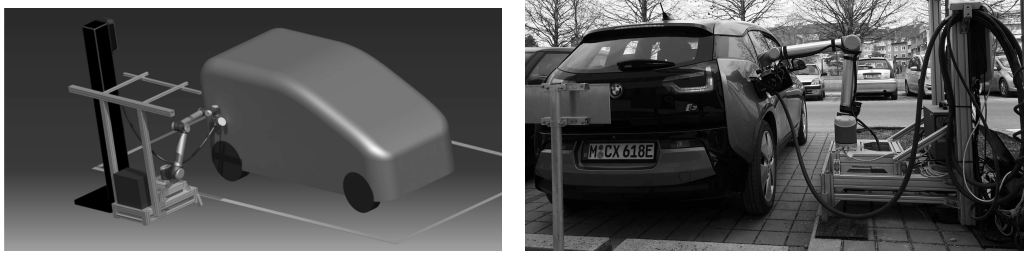
3.2.3 Paper III

3D Vision Guided Robotic Charging Station for Electric and Plug-in Hybrid Vehicles

During the exchange at *TU Graz* in Austria, a project was started to develop an autonomous robotic electric vehicle charging station. The project has targeted to solve issues becoming more relevant with a growing number of electric vehicles (EVs). One of them being many fully charged vehicles left at the charging stations and occupying the chargers, while other drivers are waiting in the queue. On top of that, more powerful fast chargers are being developed allowing the users to charge their EV for a long trip in under an hour. However, putting 120 kW or higher power requires very thick, durable and heavy power cables. For example, the weight of a 200 kW charging cable can be up to 2.26 kg/m. With longer cable lengths, this becomes difficult for people to handle, but would not be an issue for a robot [41]. The final problem comes with having many different standards for the charging sockets. It is quite common for EV owners to face a situation, where they arrive at the quick charging station to find out that the offered plug does not match the charging port of their car. In such circumstances, they either cannot charge, or have to carry some bulky adapters to ensure they can plug-in at most of the charging stations.

The project started by creating a robotic electric vehicle charging station concept. The hardware setup, robot operation, vehicle and charging port detection, the whole process work-flow and necessary communication channels between the devices together with the graphical user interface for the driver had to be considered. The concept of the charging station can be seen in Figure 3.7(a). After the initial

design of the charging station, an actual *BMW i3* vehicle was acquired for testing purposes and the charging station was constructed as shown in Figure 3.7(b). The following work focused on the algorithms for the detection of the charging port and the motion planning for the robot plugging in and out the charger.



(a) 3D design of the robotic electric vehicle charging station concept.

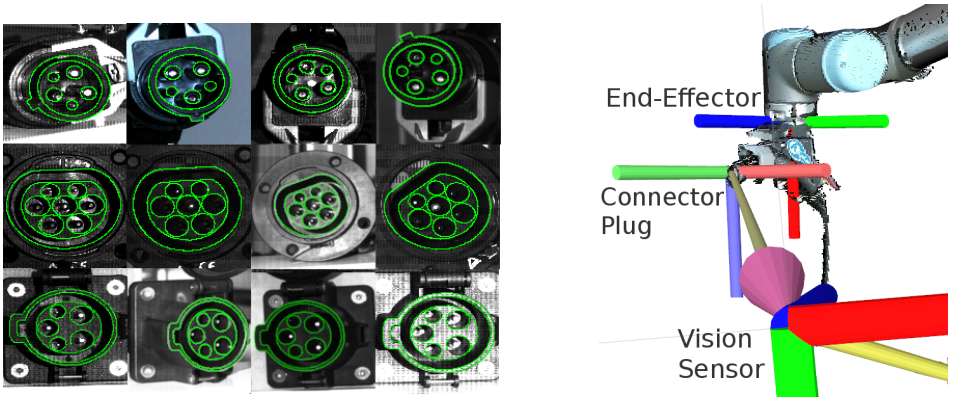
(b) The actual robotic electric vehicle charging station used for testing with a BMW i3. Image source [42]

Figure 3.7. EV charging station project, from concept to reality.

In order to allow the robot to precisely plug in the charging cable into the vehicle, an exact location and orientation of the charging port have to be determined. After testing RGB-D cameras, it was noticed that the material charging ports are made of absorbs IR light making it very difficult to get any useful depth information using these cameras. Also, the system has to be functional outdoors, where there can be infrared disturbance from the sunlight. The camera setup was shifted to use stereo cameras. Such a configuration is more robust to changing illumination conditions; however, getting depth information is more challenging.

To identify the charging port type and identify many distinct keypoints for the triangulation between the two cameras, shape-based templates were created for each of the plug types. By using template matching algorithms included in *Halcon Machine Vision* software, a precise overlay of the generated template and the image of the charging port was achieved in both camera images, as shown in Figure 3.8(a). Model matching for *Type 1* and *Type 2* charging ports as well as the connector plug (Type 2) has worked well for various illumination and angles up to 45° relative to the viewing angle of the camera. The detection distance was up to 2.5 meters, which matched the reachability of the UR10 robot. The matching confidence score for proper alignment was over 95%. The recognition speed on the full camera image was varying between $300ms$ and $800ms$. By narrowing

down the search area, for example by identifying the darker than normal regions in the image, the recognition speed can be reduced to under $150ms$.



(a) Results of the template matching. A high variety of angles and lighting conditions were tested. Viewing angles up to $45°$ resulted in successful detection with accuracy dropping beyond that. Row 1: Type 2 connector plug. Row 2: Type 1 socket. Row 3: Type 2 socket.

(b) Hand-Eye calibration results. Visualisation of the assigned coordinate frames to the vision sensor, the end-effector of the robot and the end point of the connector plug. Resulting point cloud visualising the charging plug is overlayed onto the visualisation of the robot model.

Figure 3.8. Using template matching for charging port and plug detection.

Given the accurate detection in both stereo cameras, keypoints aligned with the pins of the charging port or plug were used to calculate the depth information using triangulation. It resulted in a precise pose estimation of the charging port, which included 3D coordinates of the centre of the plug as well as orientation as roll, pitch and yaw angles.

Having an accurate detection of the charging plug, also allowed to perform a marker-less *Hand-Eye* calibration using the centre point of the connector plug as the reference point. Instead of using markers like a checkerboard for the calibration, the structure of the plug is used to get the keypoints while the robot moves it around to acquire the needed accuracy of under $1.5mm$. It becomes useful if the robot has interchangeable connector plugs of different types so that the system can re-calibrate fully automatically. Results of the successful *Hand-Eye* calibration based on a connector plug structure is visualised in Figure 3.8(b).

Once the system is calibrated and the pose of the charging port of the vehicle is detected, the three-step robot plug-in movement is initiated, as seen in Figure 3.9. Firstly, the robot moves the plug at high velocity to the approach position, which is within a 0.1 meter radius from the charging port. The second step is to reduce the

velocity to 10% of the maximum robot joint speed and move to the final alignment position. In this pose, the connector plug and the charging port are fully aligned by their Z-axis and just a few millimetres away from the contact point. The last step is to move at only 2% of the maximum speed along Z-axis and perform the plug-in motion. During this move, the forces and torques exerted on the end effector of the robot are monitored. In case the forces exceed a given threshold, the system is halted to prevent any damage.

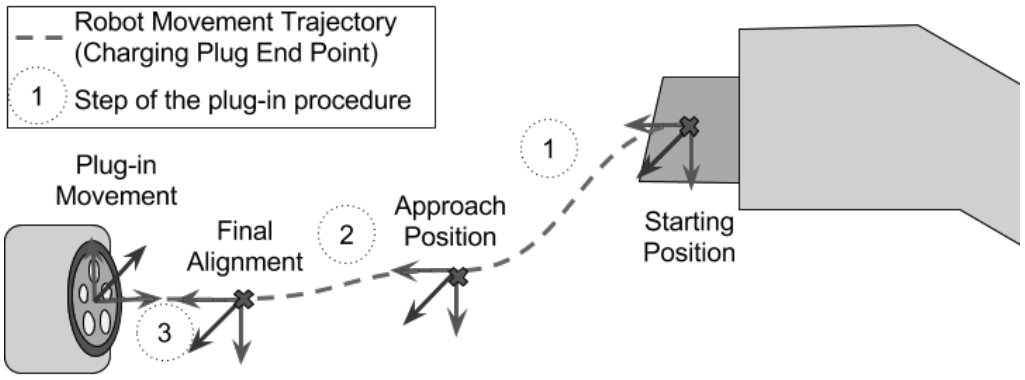


Figure 3.9. Three step plug-in procedure plan. Firstly, the robot moves the connector plug to the *Approach Position*, which lies approximately 0.1 meter away from the charging port. The second move aligns the Z-axes of the charging port and the plug and gets the plug just a few millimetres away from the port. The final plug-in movement performs the plugging in motion along Z-axis.

Under the assumption that the vehicle was stationary during the charging process when the battery is fully charged, the unplugging motion is merely the inverse of the plug-in movement by retracing the same trajectory steps in the reverse order and returning the end-effector to the standby position.

The concepts have been proven to work in 10 experiments under different rotation angles that the plug was placed at. It has worked successfully 8 out of 10 times, with it failing twice due to the rotation misalignment. Even with small angular offsets of up to 5°, the plug was inserted successfully making contact.

The work has resulted in the actual working robotic electric vehicle charging station, where our approach has proven to work under varying lighting conditions. The project has been successfully submitted to the partners and was released publicly in September 2018 followed by some mainstream media articles about it [42]. Furthermore, a patent citing this work was published in 2019 by Intel Corporation [43].

3.2.4 Paper IV

Robot Localisation and 3D Position Estimation Using a Free-Moving Camera and Cascaded Convolutional Neural Networks

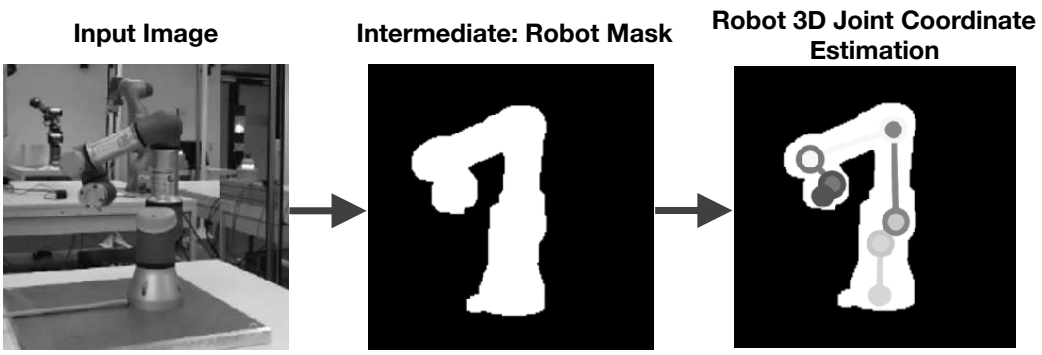
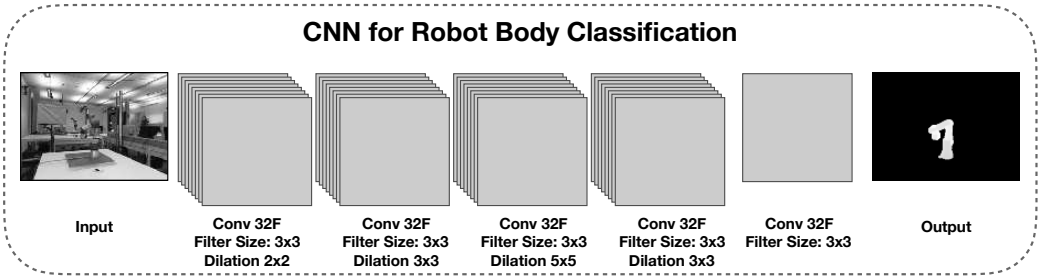


Figure 3.10. A simple colour image of the robot body is provided as an input to the system. The first CNN estimates the mask containing the robot body and this result is overlayed with the colour image and used as an input to the second CNN. The second CNN provides an estimate of the joint coordinates of the robot in 3D. Each robot joint is visualised with a circle of a different colour.

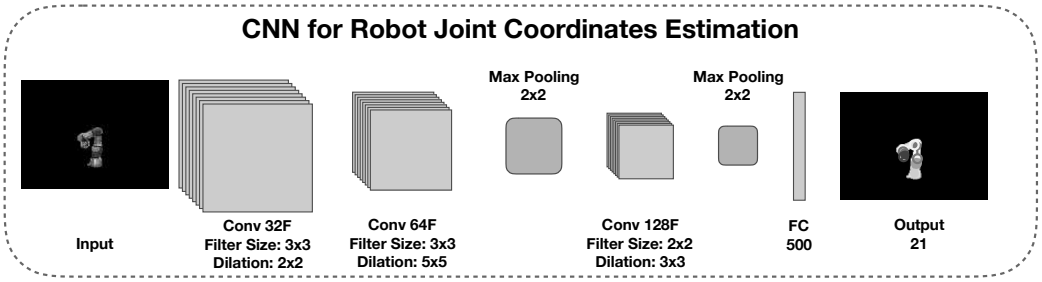
Each of the robot-camera system in this thesis so far required performing a *Hand-Eye* calibration. It was not only time consuming, but it also had to be done every time the camera or the robot was moved in relation to each other. A new challenge was set to find a workaround where calibration would not be necessary anymore. Given the recent advances in deep learning based object recognition for item classification or self-driving applications, using convolutional neural networks (CNNs) for this purpose was analysed. Robot arms have distinct visual features, and humans can easily identify robots in environments like factory floors.

The first part of the cascaded CNN is used to identify the mask of the robot in the input image. Then the estimated mask is overlayed with an input image and fed as an input to the second layer of the cascaded CNN, which estimates the position of each of the robot joint in 3D relative to the camera. The whole process is summarised in Figure 3.10.

The work started by collecting training datasets for the full line of Universal Robots: UR3, UR5 and UR10. Kinect V2 cameras were used as vision sensors with the robot joint encoder information used to precisely fit the robot model in the recorded image, both colour and depth. Depth information was only used to create training dataset, while only a 2D colour image is needed as an input to the system. This information could be used to determine the mask of where the robot is located in each image. Furthermore, 3D positions of each of the robot joints



(a) CNN architecture for the robot mask classification. The network consists of 5 convolutional layers with varying dilation. Input is a colour image and output is a mask image defining the body of the robot.



(b) CNN architecture for the robot joint coordinate estimation. The network consists of 3 convolutional layers, 2 pooling layers and a fully connected layer in the end. Input is an overlayed colour image with a robot foreground mask and output is 3D coordinates of the robot joints in the coordinate system of the vision sensor.

Figure 3.11. Cascaded CNN architecture for the robot position estimation.

were saved for each of the frames. For each of three recordings, the camera was placed at many positions overlooking the robot with varying angles and distances resulting in different backgrounds. During each of the recording, the robot was moved in many different configurations to add a variety of viewing angles. Dataset summary can be found in Table 3.1.

The training datasets were then used to train a specially designed cascaded CNN to optimise for the following objectives:

- The robot mask in the input image
- 3D coordinates of each of the robot joints in relation to the camera

Despite the depth information being used during the training, an input to the CNN for the detection phase was a simple 2D colour image. The structure of the network can be seen in Figure 3.11.

Results of the system were promising with the mask estimation accuracy between 92.8% and 98.1% depending on the robot type, and the joint position esti-

Table 3.1. Dataset summary describing the number of samples collected for each type of the robot. In total 9 recordings were made, 3 for each type of the robot.

Recording	Robot Type	Number of Frames
Rec 1	UR3	211
Rec 2	UR3	252
Rec 3	UR3	463
Rec 4	UR5	252
Rec 5	UR5	756
Rec 6	UR5	1512
Rec 7	UR10	112
Rec 8	UR10	278
Rec 9	UR10	514

mation error was under $4cm$ in all the cases with as low as $2.02cm$ error for the UR5 robot.

The work resulted in a cascaded CNN based system capable of successfully finding the robot in a 2D colour image and estimating its 3D base and joint coordinates with an accuracy of approximately 3 cm compared to the ground truth.

3.2.5 Paper V

Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images

Given the promising results of the previous work using cascaded CNNs described in Section 3.2.4, the main problems remaining with that system were addressed in *Paper V*. Having the cascaded structure, each network had to be trained separately for each of the robot types, meaning the system was not universal. Furthermore, the training process was split into two for each of the objectives, and it was difficult to determine the optimum configuration that would minimise the error both for the mask and for the coordinate estimation.

It motivated a reconsideration of the structure of the deep neural network used and finding a more versatile solution to the problem. Given the three robot types used and multiple objectives to optimise the system for, a new approach of using a multi-objective CNN was found. It can tune for four goals simultaneously: Robot Mask, Robot Base Coordinates, Robot Joint Coordinates and Robot Type. This information is sufficient not only for the robot detection and pose estimation in an un-calibrated system, but also to classify *which* robot is seen by the camera.

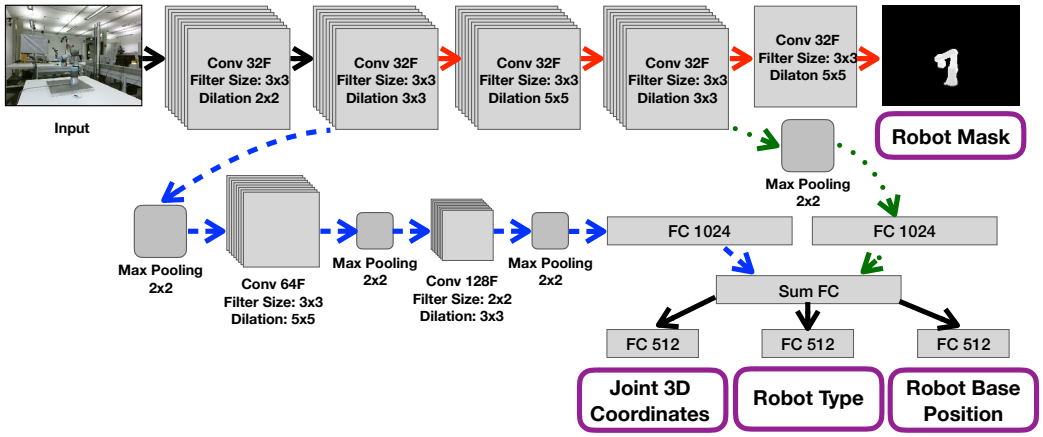
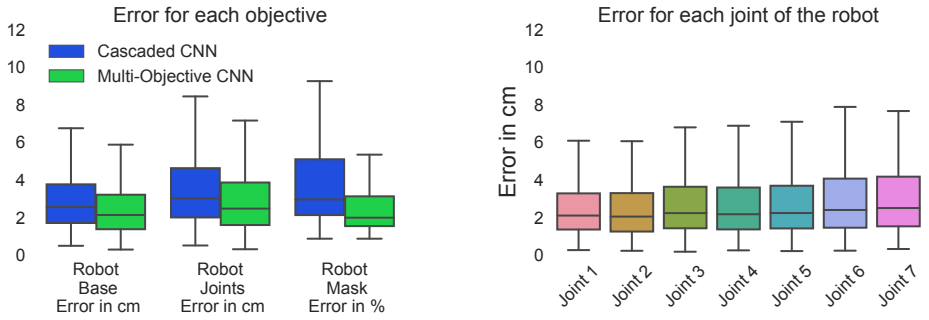


Figure 3.12. Multi-objective CNN structure. Input is a simple 2D colour image, and the network is trained for four outputs: robot mask, 3D coordinates of robot joints, 3D coordinates of robot base position and robot type. There are two main branches of the CNN. The first one is aimed to learn the features leading to an accurate robot mask mainly consisting of dilated convolutional layers, marked by solid red arrows. The second branch, marked in blue dashed arrows, consists of max pooling and dilated convolutional layers with fully connected layers at the end. The whole CNN is trained for all four outputs simultaneously using a common loss function.

To allow for a direct comparison with the previous method, the same training and test datasets were used to evaluate the multi-objective CNN approach. The network itself has multiple branches splitting and joining to share information in some of the layers and reduce redundancy. The network structure is seen in Figure 3.12. Given that the whole network is trained simultaneously, a single loss function had to be constructed optimising for all of the objectives. It was solved by using a weighted sum of the loss functions for each of the goals.

The results of the multi-objective CNN approach compared to the cascaded CNN method were promising. The system not only could be trained once for all the robots but also improved accuracy on all of the objectives. The comparison between the two methods can be seen in Figure 3.13(a) and error of the joint coordinates in Figure 3.13(b). Furthermore, the detection time or feed-forward of the multi-objective CNN was just $15ms$ on a GPU, proving that the method is capable of real-time detection in a live image stream.

The work resulted in a successfully working multi-objective CNN, which can estimate the robot mask and estimate the coordinates of its base and joint positions using just a 2D colour image as an input. Results have shown an improvement in all the detection metrics compared to the previous cascaded CNN approach.



(a) Comparison of the Multi-Objective CNN approach against the Cascaded CNN approach [44]. Errors for all the objectives are smaller, as well as the range of quartile values.

(b) Error for the 3D coordinate estimation for positions of each robot joint. It can be seen that the error slightly increases for joints further away from the base.

Figure 3.13. Evaluation of our method by testing the trained system on the test dataset.

3.2.6 Paper VI

Transfer Learning for Unseen Robot Detection and Joint Estimation on a Multi-Objective Convolutional Neural Network

Having a common multi-objective CNN capable of training for all of the desired outputs simultaneously added a significant amount of flexibility to the system. However, it was still limited to the robots produced by Universal Robots and required collecting vast amounts of training data. These two main issues were addressed in *paper VI*.

To test the ability to adopt a readily-trained multi-objective CNN to a new, and significantly different robot type, a Kuka iiwa LBR robot was applied. Not only does it have a different appearance, but it also has seven degrees-of-freedom (DoF) compared to 6 DoF of all of the Universal Robots. A training dataset was similarly collected for the Kuka robot as it was done before for the UR robots.

However, instead of re-training the system from scratch, a fully-trained system from our previous work was taken as a starting point. Then, some layers in the multi-objective CNN were frozen, which means that during the training process, the parameters of those layers are not modified, while the remaining layers are being trained. It allows to reuse more general visual features learned by the network, and outer layers adapt how to utilise them to identify a new type of the robot. Furthermore, an additional output for the 7th joint of the robot was added. The network structure and frozen/unfrozen layers can be seen in Figure 3.14. Essentially, it is the same multi-objective CNN structure as in work described in

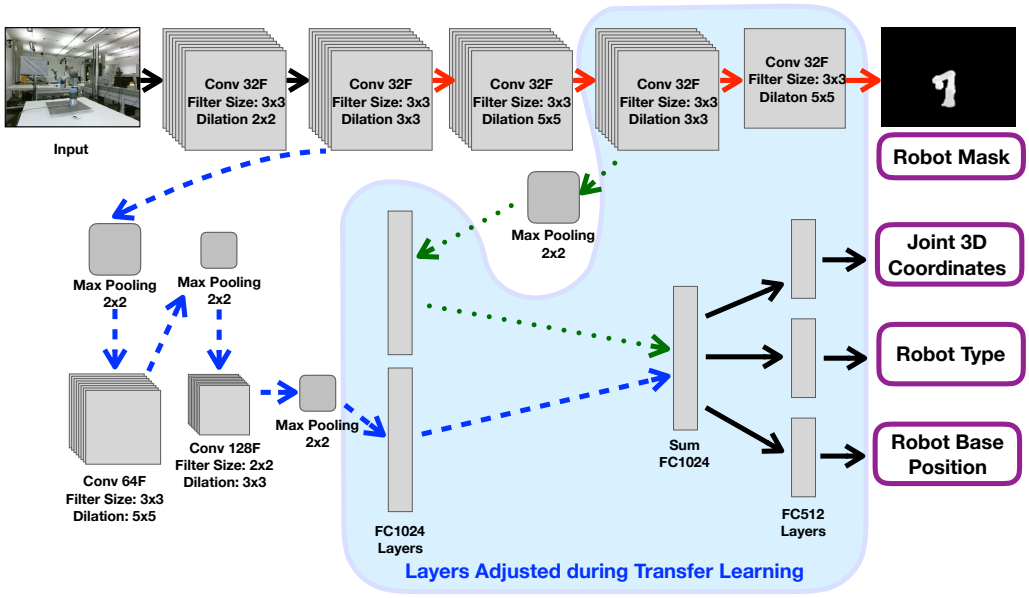


Figure 3.14. Structure of the multi-objective CNN. The network uses part of common convolutional layers and then branches off for objective-specific training. Fully-connected (FC) layers, marked by the blue area, are the ones being adjusted during the transfer learning process to adapt to the new robot model. Convolutional layers learn generalised visual features of the image, so their parameters stay *frozen* during the transfer learning. It allows for quicker adaptation with a limited number of input images compared to the full training process.

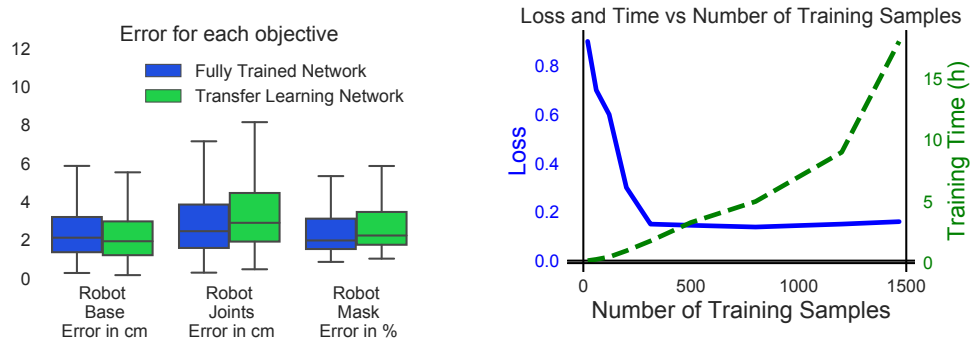
Section 3.2.5.

The results of the transfer learning approach have proven the capability of the system to adapt to the robot using a limited amount of training data, thus significantly reducing the training time. While the full training of the network took around 60 hours for the UR robots, the transfer learning approach to adapt to the Kuka robot took only around 2 hours on the same GPU hardware.

Furthermore, a detection accuracy almost equal to the fully trained system could be reached with just using 300 training samples compared to over 2000 needed for full training of the system. Performance results are shown in Figure 3.15.

This work has moved the previously-shown concept of using a multi-objective CNN to find the robot and estimate its position in 3D space a lot closer to the real-world applications by reducing the need for large amounts of training data to adapt it to new robot types.

The work resulted in significantly shorter training time and a smaller training dataset by using a transfer learning method on an already trained multi-objective



(a) Comparison of a fully-trained network on UR robots [45] with a transfer learning method on Kuka. Error slightly increased for joints position estimation and robot mask, however, transfer learning method was somewhat more accurate in estimating the position of the robot base.

(b) Loss function and training time against the number of training samples used. It was acquired by running experiments using input datasets of different size. Using more than 300 training samples does not give accuracy benefit, while just increasing the training time.

Figure 3.15. Evaluation of the transfer learning method using the test dataset.

CNN to adapt it to a new robot type.

3.2.7 Paper VII

Two-Stage Transfer Learning for Heterogeneous Robot Detection and 3D Joint Position Estimation in a 2D Camera Image using CNN

Using the transfer learning approach to re-train a multi-objective CNN to a new robot type was an efficient method. However, the problem remained that the network was adapted to a new robot and *forgot* the previously-trained robot types. In this work, we develop a method to include new robot types into the multi-objective CNN without forgetting previously-trained information, making it capable of recognising all the robots at once.

A new robot was added: Franka Emika Panda. It has 7 DoF and plain white-black appearance, which is even more difficult to distinguish in front of white or grey walls, making it a more difficult challenge. The training dataset consisted of a collection of frames containing all the robots: Universal Robots, Kuka and Franka Panda. It allowed the network to train to include a new robot type, while still being able to recognise previously-known robots.

The transfer learning approach was modified to a two-stage model, shown in Figure 3.16. Stage 1 adjustment consisted of the outer layers of the CNN, and the training was continued until no more improvement in the loss was detected. Once

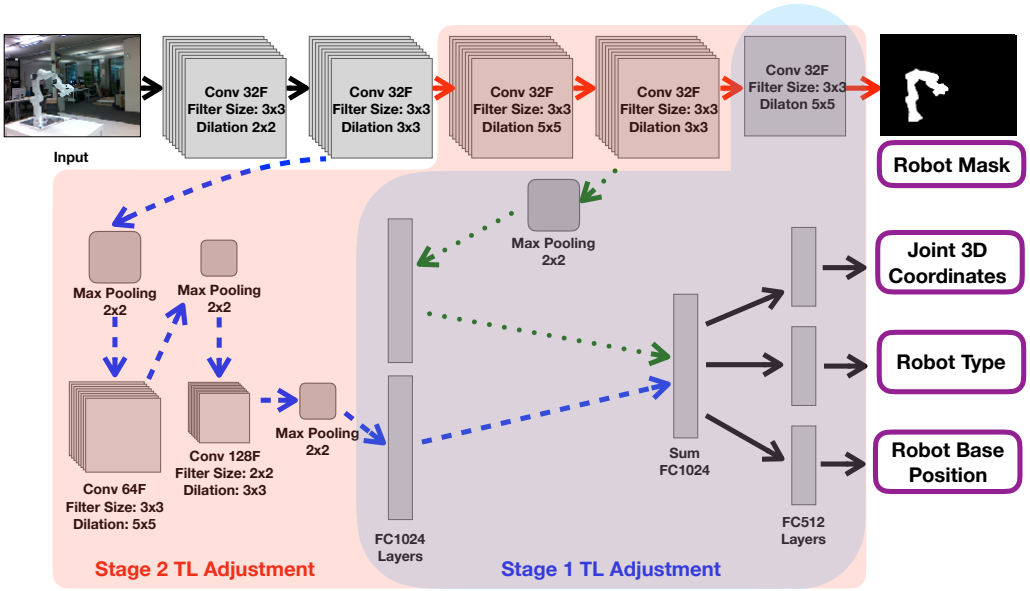
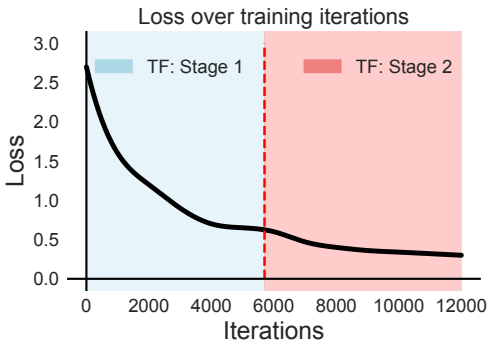


Figure 3.16. The multi-objective CNN with a two-stage transfer learning. The network is taught in two stages using the transfer learning approach. In stage 1, the parameters for all the layers, besides the final ones marked in blue are *frozen* and the system is trained until there is no more improvement. Afterwards, in stage 2, the parameters of the CNN layers marked in red, together with all the stage 1 layers, are adjusted during the training.

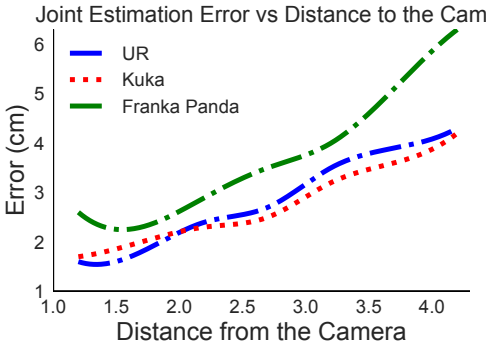
the training settles, Stage 2 starts where more layers are unlocked, and the neural network trains further to improve the detection accuracy. This approach allows to keep the training process still short but adjusts more parameters in the CNN to adapt it to more distinct visual appearances of the robots.

A new training dataset collection method was developed to avoid overfitting. By adding motion capture markers on the camera, we were able to precisely track the vision sensor and move it freely around the space capturing the robot in a large variety of backgrounds and changing lighting conditions. This more diverse data ensures that the CNN is learning the right features of a robot instead of figuring out how to identify the background, which would prove difficult recognition in real-world scenarios.

Compared to the previously presented work in Paper VI, the detection accuracy of the current two-stage transfer learning approach achieved similar accuracy with a joint position error of $2.46cm$ vs $3.12cm$ and slightly worse accuracy for a robot mask estimation: 97% vs 98% in the previous work. Full training time of the multi-objective CNN for UR robots took 60 hours vs 10 hours in the current work to include the detection of Kuka and Franka Panda robots based on *two-stage*



(a) Loss function against the number of training iterations. Stage 1 and stage 2 of our two-stage transfer learning approach are marked by background colours on the graph.



(b) Errors for 3D position estimation of robot joints depending on the camera distance from the robot. Results are grouped by the robot type.

Figure 3.17. Evaluation of the two-stage transfer learning method using the test dataset.

transfer learning.

Furthermore, a more precise analysis was done on the detection precision depending on the distance between the camera and the robot, visualised in Figure 3.17(b). There is close to a linear relationship between the distance between the robot and the accuracy of the 3D position estimation of the robot joints. The performance of each training stage of transfer learning is shown in Figure 3.17(a). Stage 1, where the parameters in final CNN layers are adjusted, saturates after 6000 iterations. Afterwards, further layers are *unlocked* switching to Stage 2, and the loss function reduces even further settling down between $10000 - 12000$ iterations.

The work resulted in the successful expansion of the multi-objective CNN to include two new robot types: Kuka and Franka Panda, to an already existing CNN trained on Universal Robots, without forgetting previously-trained information. Camera movement was fully unconstrained and moved by hand while being tracked by a motion capture system. It is not necessary, but it allows for collecting a significantly more diverse training data resulting in a more robust system. A two-stage transfer learning approach ensures rapid learning to include a new robot type by having a fraction of the training data compared to a full training process.

Chapter 4

Discussion

This section discusses the work presented in this thesis. First, the research approaches taken throughout the work together with the achievements are discussed in Section 4.1. Limitations of the given work are described in Section 4.2.

4.1 Research Approaches and Achievements

When addressing the *research questions* of this thesis, some decisions regarding the research approaches were made. All the hardware used consisted of standard commercially-available products without any hardware modifications. It provided many benefits: the equipment could be easily swapped with new components, and the system could be quickly expanded to have more cameras or robots. Furthermore, the cost of the equipment is significantly lower compared to the custom designed robots; additionally, the commercially-available hardware is already certified for use with people. The discussion below is divided into two sections addressing each of the *research question* previously defined in Section 1.2.

4.1.1 Collision-Free Robot Operation in a Shared Workspace

Initially, the goal was to operate the system indoors in a fixed environment, where the robot and the surrounding sensors would have dedicated fixed locations, allowing for high-accuracy visual detection of objects that are close to the robot. The research focus was oriented to the mapping of the workspace, making the system immune to visual obstructions and developing the robot behaviour model allowing for an adaptive and collision-free operation in a dynamic environment.

The robot control model was developed for higher-level commands, like trajectory generation in Cartesian or joint coordinates with the actual execution of the movement completed by using *ROS* and *MoveIt!* libraries and drivers to calculate inverse kinematics and motor commands. Given the movement requirements, the robot reaction and movement time were sufficient for the given applications, as tasks requiring high-frequency feedback, like force control, were not necessary. Quick robot reactions to unexpectedly-changing conditions could be achieved by reusing previously-calculated trajectory points instead of starting new calculations. It was proven in a reflexive behaviour model developed in work described in Paper II.

Many iterations of testing different vision sensors were done. They varied from testing regular cameras to *RGB-D* cameras, where the latter provides depth information on top of the colour image. Despite the differences, an *Eye-to-Hand* calibration had to be performed for every new setup with a new camera or a new configuration. This lead to the development of the automatic calibration system, where the robot was used to move the calibration board and self-calibrate together with the vision sensors connected to the system, described in Paper I. Furthermore, it was noticed that some materials, like rough black matt plastic, typically used in car manufacturing for parts like charging ports on electric vehicles, highly absorb infrared light. It led to noisy and inaccurate measurements of any *RGB-D* cameras, which use infrared patterns to calculate the depth information. A workaround was to use a stereo camera setup instead, which could provide depth information on such difficult materials as well by using shape-based methods at the cost of more expensive hardware. However, a stereo camera setup performs better in an outdoors environment, where infrared noise is introduced by the sun. Given that the work described in Paper III was aimed at charging electric vehicles, it is necessary for the system to work both indoors and outdoors.

Referring back to the defined research question 1 (RQ 1), the system was able to ensure collision-free operation in a dynamic environment. The robot was capable of reacting quickly enough to avoid even fast-moving obstacles. However, the robot was keeping a safe distance of at least 2 cm from obstacles to compensate for the errors in detection and workspace mapping.

4.1.2 Robot Body and Pose Estimation Using Deep Learning

With the following work, the research focus shifted to machine learning approaches for robot-camera systems. In previously-presented solutions, the system highly

relied on having uninterrupted communication channels to send camera images, real-time robot position and even relied on an exact robot model information. If any of this information were delayed or inaccurate, the whole system would possibly malfunction. To reduce the high dependency on multiple systems working reliably, deep learning approaches were used to teach the computer to recognise the robot in the image and estimate the exact position where it is located in 3D space without having any direct information from the robot sensors or even having an explicit model of the robot.

Given the task requirements of using a 2D colour image as an input and predicting a mask of the robot in an image as well as the 3D position of each joint of the robot, convolutional neural networks (CNNs) were chosen as the most suitable approach to solve the task. CNNs are capable of learning complex tasks by effectively adjusting large amounts of weights between neurons and selecting the most appropriate filters automatically. However, they require extensive training datasets with correctly marked ground truth to train efficiently. An automatic training data collection system was set up using Universal Robots and RGB-D cameras, which allowed to collect thousands of training samples, which could be used to train the CNN.

The first successful approach was by using cascaded CNNs, where the first CNN was used to estimate a mask of the robot in the image and then the result was passed on to the second CNN, which predicted the 3D position of the robot joints, as described in *Paper IV*. The system has proven to work with a few-centimetre accuracy. However, the training process was cumbersome. A separate CNN was necessary for each of the robot types, and if the robot mask estimation had significant errors, the joint position estimation was prone to fail because of imperfect input.

Naturally, a better CNN structure was explored to make the system more robust. Instead of passing output from one CNN to be used as an input in the further process, the structure of the CNN was modified by merging both CNNs into one. It resulted in a multi-objective convolutional neural network, which was capable of estimating four outputs at the same time: robot type, robot mask, 3D base position of the robot and 3D positions of robot joints. The improvements meant that the system was able to not only estimate all the same information as cascaded CNNs but also identify the robot type. Everything was done in a single neural network. The improved system resulted in *Paper V*. On the other hand, larger multi-objective CNN meant that teaching it to recognise all the robots required training data for all of them to be collected and used in the training process all at

once.

Despite having an automatic collection method for the robot training data, it was still a time-consuming process. The robot had to do hundreds of physical movements, with the camera frames captured and saved for each of the movement. In some cases, there were still some unexpected irregularities or accuracy issues present when the trained system was tested in operation. Even though a multi-objective CNN was capable of recognising multiple robots, once the system was trained, it was limited just to the robots it was taught. If a new robot was added, the whole training process had to be repeated. A new approach was explored on how to reduce the amount of training data and time needed for adding a new robot to the recognition system. *Transfer learning* can re-train the network for a new robot type using a limited amount of training data. It allowed to reuse low-level features and reduced the amount of needed training data ten-fold as well as cut the training time from 60 to 2 hours. The work was described in *Paper VI*.

Re-training CNN using *transfer learning* is very efficient, but one significant issue remained. CNN effectively learned to recognise the new robot. However, it *forgot* the previously-known robots and was unable to identify or estimate their position anymore. It would not be useful in practical applications. Aiming to solve this problem, the transfer learning approach was adjusted not only to teach CNN to identify the new robot but also use the information of previously-known robots in the training process. This way, the network is capable of including the new robot into the current recognition system without losing previous knowledge. Given the more complicated setup, a *two-stage transfer learning* approach was used, where a number of layers to be adjusted was changing during the training process to allow the system to adapt better, but still keep the training time low. Eventually, the system was capable of using the base CNN trained on *Universal Robots* and include both *Kuka* and *Franka Panda* robot recognition in the same multi-objective CNN by using a two-stage transfer learning method presented in Paper VII.

Referring back to research question 2 (RQ 2), we have been able to teach the system to recognise the robot not only in an input image but also to estimate its position in 3D space in relation to the camera. It was made possible by using deep learning and CNNs. It allows the robot to be recognised under various illumination or in dynamically re-configurable camera-robot environment. To train the network having just a limited amount of training data, a readily-trained network can be taken with the transfer learning approach applied to it to incorporate recognition of new robot types.

4.2 Limitations

All the work in this thesis was based on using the actual hardware, both for robot platforms and vision sensors. It was a significant complication compared to simulation-based approaches. Problems to address ranged from sensor noise, which is hard to model, to constantly-changing conditions of the real world, some of which were initially known, while others unexpectedly appeared when conducting experiments and evaluating the system. In this section known limitations will be discussed by dividing them into three main categories: robot limitations, vision sensor limitations and deep learning limitations.

Some of the limitations, especially hardware-based ones, are out of control by the researchers. However, most of the known limitations were approached during the development of the presented methods to minimise their impact on the performance of the system.

4.2.1 Robot Limitations

Collaborative robot arms have high flexibility and mostly intuitive control, usually with a collision detection method. In this thesis, it was decided to use standard robot controllers through a *ROS* interface to control the robots. It allowed for rapid integration and effective robot control; however, at the same time, introduced some problems. First of all, the actual high-frequency real-time control of the robot was not possible using both standard and modified *ROS* interface with the robot. The refresh rate of the controller was too low for real-time control. Thus, approaches like force-feedback were not possible and not explored. At the same time, throughout the work, some random delays were experienced, which were at the robot controller level and difficult to compensate. Using this approach, the researcher has to rely on the good functionality of the algorithms provided by the robot manufacturer as well as path planning algorithms provided by *ROS* and *MoveIt!* packages. An alternative approach could be taken by controlling the robot motors directly. However, that would require developing its own implementation of the robot's inverse kinematics and control algorithms, taking up a significant amount of time focusing purely on engineering tasks.

4.2.2 Vision Sensor Limitations

A new low-cost commercially-available RGB-D sensor is a great choice when not only a colour image but also depth information is needed. Creating models of the

robot workspace was made significantly easier by having distance information or even full point clouds of the area. However, this meant a significant increase in the sensor data flow. Typically, these sensors required *USB 3* connection to the computer, and the amount of data to be processed used most of the resources of the computer, leaving little processing power to the rest of the algorithms, like robot trajectory planning. Some optimisations were taken, but having more than three sensors connected to one machine was complicated to get to work.

Furthermore, having *structured light* based sensors, like *Kinect V1*, meant that multiple sensors observing the same object would have their projected IR patterns interfere with each other, significantly reducing the accuracy or failing to provide depth data overall. Newer sensors based on *time-of-flight* approach had less or no interference allowing to use multiple of them overlooking the same area at once. However, the impact of having strong infrared light sources around, like a motion capture system with active IR lights or direct sunlight still highly reduced the depth data accuracy.

Given that most of the RGB-D sensors use a combination of projecting IR patterns and using stereo matching, there is a distance limitation, both minimum distance the object can be detected at, as well as maximum distance. The successful depth detection for an RGB-D sensor is typically in the range from 0.3 meters to 10 meters depending on the sensor. It has to be accounted for when operating the system, as getting too close to the camera means that objects will not be detected anymore.

Occlusions is another major issue in robot workspace mapping tasks. Having multiple cameras can add redundancy, but still, does not guarantee entirely reliable detection. If a large object is right in front of the camera, there is no coverage of what is happening behind it. As previously mentioned, having too many cameras is complicated because of the price and large amounts of data coming from each of the sensors. The placement to give full coverage is crucial to reduce the chance of occlusions.

Commercially-available sensors provide an excellent quality-to-price ratio. However many cheaper and consumer level RGB-D sensors, like the ones used in this thesis, have limited support and might have reliability issues. It is common to have some of the products cancelled without prior notice or not receive any help when the product is difficult to integrate or starts malfunctioning.

4.2.3 Deep Learning Limitations

Deep learning provides great adaptability compared to some of the probabilistic methods, but at the same time, it is difficult to guarantee reliable functionality. Usually, deep learning based approaches can be evaluated using created test cases and datasets, however, it is hard to analyse analytically or genuinely understand how the network has learned to solve the task and how internal parameters were set.

The quality of the system highly relies on the training data and loss functions, which were provided. It might work correctly in the given conditions, but when some changes occur, even what might seem non-significant for a human, the system can suddenly malfunction without any prior warning. In many cases, this is due to overfitting. It happened when the training data was not diverse enough to cover all the possible scenarios where the system will have to operate. It learns specific features instead of an overall understanding of the objects. Overfitting can be complicated to recognise by a researcher, as no indications are given during the training process. Even with validation set having unseen data, a deep learning network might learn general background features and provide good accuracy. A significantly different validation set is needed to provide a robust measure of network performance.

Throughout the work, it was a challenge to create a dataset containing a large variety of illumination conditions, record the robot in front of different backgrounds and make sure the data includes as many different robot position configurations as possible. Furthermore, it was not straightforward to find good weight values for the multi-objective loss function, which consists of four loss functions combined.

4.3 Conclusions

This thesis has contributed to creating an environment-aware robotic manipulator system by combining a collaborative robotic arm with commercially-available low-cost vision sensors. Algorithms were developed allowing the system to identify the robot arm in a simple 2D colour image, map the workspace of the robot and detect obstacles appearing close to the robot. Furthermore, this visual information was used to control the robot to allow the robot to operate safely in a dynamic environment by avoiding any of the obstacles in real-time. Application of these methods in a real-world setting was proven by the development of

the robotic electric vehicle charging station capable of autonomously plugging in and plugging out an electric vehicle parked in front of it. As far as authors are aware, this has been the first research publication of a robot-based electric vehicle charging station based on vision-guided robot movements.

The main contributions of the thesis can be summarised as follows:

- The automatic *Hand-Eye* calibration method for camera-robot systems has introduced a quick approach to calibrating multiple cameras to the robot with minimal manual work. It allows for a significantly faster reconfiguration and setup of such systems, while the accuracy achieved matches the traditional manual calibration methods.

- Modelling the workspace of the robot as an octomap and constructing a danger map has been proven as a robust method to incorporate point clouds from multiple 3D cameras and still allow for real-time performance. This map can be used to plan the robot motions by finding the optimum path considering the length of the trajectory and the risk of colliding with an obstacle by using historical observation data.

- The predictive-reflexive robot behaviour model allows for safe robot motion planning in a dynamic environment with a fail-safe reflexive motion enabling the robot to move away from unexpected obstacles quickly. During the reflexive movement, a new trajectory is recalculated for the further safe operation of the robot.

- Using a stereo camera setup in detecting the shape of objects produced from materials absorbing and dispersing infrared light has been proven to be a viable alternative in situations, where IR based 3D cameras fail. Furthermore, it is more robust in outdoor operations, where sunlight can cause interference. Shape-based detection and pose estimation was used to successfully find the charging ports of real electric vehicles and use multi-stage motions to plug-in and plug-out the charging cable to the car. The approach has been proven to work in a real-life scenario.

- Cascaded convolutional neural networks were successfully applied to robot detection in the systems as an alternative to *Hand-Eye* calibration. Instead of using both colour and depth information from the 3D camera, an input is limited to just a simple colour image. The robot body in the picture is precisely identified as well as 3D positions of the robot joints were estimated within an accuracy of a few centimetres.

- A multi-objective CNN was created to add more flexibility to the system and combine the estimation of both the mask of the robot as well as 3D coordinates of the robot joints. A single training process can be used to optimise the neural network for all of the objectives at once instead of training each of the CNNs separately in the cascaded structure.

- A transfer learning approach has reduced the training time as well as the amount of the training data needed, while still achieving a comparable accuracy to the fully-trained systems. It allowed us to take a multi-objective CNN trained on one robot type and adapt it to a new robot type significantly faster compared to the full-training cycle. Furthermore, a two-stage transfer learning approach was used not only to adjust the neural network but also to incorporate the detection of multiple robots all at once. Eventually, the system has been proven to work in detecting robots from three different manufacturers with significantly different visual appearance: Universal Robots, Kuka iiwa LBR and Franka Emika Panda.

Given these results, we believe that the achieved results have proven the capability and potential of the system. It can act as a suitable platform for further development of advanced collaborative robot systems, which is taking inspiration from the understanding of an environment similar to what can be seen in humans and animals. Furthermore, the next level of safety and collaboration could be achieved by improving these methods further.

4.4 Future Work

The future work will focus on improving the performance and robustness of the developed CNN approach, and testing it in real-world applications.

One of the objectives is to analyse the parameters of the multi-objective CNN to figure out the optimum weights for the loss functions, the input size of the images and revise the structure of the CNN itself. It would provide a better in-depth understanding of how CNN is learning and give inspiration to ideas on how the process could be improved. Weight adjustment would make sure that enough emphasis is placed on each of the objectives that the CNN is trained on and find the optimum balance of accuracy between the mask and robot joint coordinate estimation. The CNN structure could be further optimised in at least two ways. The first is to reduce the size of the model, so it can train and run faster and use less

memory on the GPU. The optimisation would be aimed at reducing the number of parameters; thus, the number or size of the layers in the network. Another optimisation could be targeted at improving accuracy by changing the structure. One considered approach of that would be to combine the research conducted in our lab and apply evolutionary algorithms for these adjustments. However, it is a very computationally-expensive and time-consuming process. On the other hand, given a fully automatic method to explore various network configurations, it could be done autonomously without any manual intervention.

Collecting more diverse training data would allow making the CNN system significantly more robust. At the moment, all the training data was obtained in the labs of the following three institutions: University of Oslo, TU Graz and Joanneum Research: Robotics Institute. To apply this system for real-world scenarios, a significant amount of training data should be collected in the environments where the system could be used, such as, but not limited to hospitals and surgery theatres, robotic charging station, collaborative robot setups, warehouses and factory floors. A flexible data collection approach has already been developed by using optical tracking systems (Optitrack) together with the robot self-filtering algorithms to automatically mark the ground truth for the training datasets during the collection process. However, such a system is not easily portable so that alternative tracking approaches could be considered.

So far, the environment understanding was limited to vision. However, more senses could be incorporated. These include force sensing, proximity sensing using ultrasound sensors or laser scanners, incorporation of internal torque sensors, as well as sound. All of this data provides significant amounts of information in the biological world. Thus, it can be interesting to add it to our system as well. We believe that it could allow the robot to react to dangers, which could not be detected visually, but can still cause dangerous situations. For example, if an object, which is already in contact with the robot, starts exerting high amounts of force, which could cause damage to the hardware.

I

Chapter 5

Papers

This Chapter contains a collection of papers published in this PhD thesis.

PAPER I

Automatic calibration of a robot manipulator and multi 3D camera system
J. Miseikis, K. Glette, O. J. Elle and J. Torresen
2016 IEEE/SICE International Symposium on System Integration (SII), Sapporo, 2016, pp. 735-741.

Automatic Calibration of a Robot Manipulator and Multi 3D Camera System

Justinas Mišeikis[1], Kyrre Glette[2], Ole Jakob Elle[3], Jim Torresen[4]

***Abstract*— With 3D sensing becoming cheaper, environment-aware and visually-guided robot arms capable of safely working in collaboration with humans will become common. However, a reliable calibration is needed, both for camera internal calibration, as well as Eye-to-Hand calibration, to make sure the whole system functions correctly. We present a framework, using a novel combination of well proven methods, allowing a quick automatic calibration for the integration of systems consisting of the robot and a varying number of 3D cameras by using a standard checkerboard calibration grid. Our approach allows a quick camera-to-robot recalibration after any changes to the setup, for example when cameras or robot have been repositioned. Modular design of the system ensures flexibility regarding a number of sensors used as well as different hardware choices. The framework has been proven to work by practical experiments to analyze the quality of the calibration versus the number of positions of the checkerboard used for each of the calibration procedures.**

I. INTRODUCTION

In many practical applications, industrial robots are still working "blind" with hard-coded trajectories. This results in the workspace for robots and humans being strictly divided in order to avoid any accidents, which, unfortunately, sometimes still occur. Furthermore, working in a dynamic environment without a direct connection to other machinery sharing the same workspace, might prove difficult. It is often more common to have collision detection systems, which do not always work as expected, rather than collision prevention methods [1]. However, *environment-aware robots* [2] [3] are becoming more common, both developed in research and by robot manufacturers themselves, e.g. Baxter by Rethink Robotics [4].

Low-cost and high-accuracy 3D cameras, also called RGB-D sensors, like Kinect V1 and V2 [5], are already available. They are suitable for a precise environment sensing in the workspace of a robot, providing both color image and depth information [6]. However, external sensors are commonly used in *fixed positions* around the robot and are normally not allowed to be moved. After any reconfiguration in the setup, the whole system has to be calibrated, usually by a skilled engineer. Camera calibration can be divided into two main stages:

- Internal camera parameters, like lens distortion, focal length, optical center, and for RGB-D cameras, color and depth image offsets [7] [5].

[1 2 3 4]Justinas Mišeikis, Kyrre Glette, Ole Jakob Elle and Jim Torresen are with the Department of Informatics, University of Oslo, Oslo, Norway
[1 2 4] {justinm,kyrrehg,jimtoer}@ifi.uio.no
[3]Ole Jakob Elle has his main affiliation with The Intervention Centre, Oslo University Hospital, Oslo, Norway oelle@ous-hf.no

- External camera parameters: the pose (position and orientation) of a camera in a reference coordinate frame. It is commonly called Eye-to-Hand calibration [8] [9]. The Eye-to-Hand calibration, or transformation from the camera coordinate system to the robot base coordinate system is shown in Figure 1.

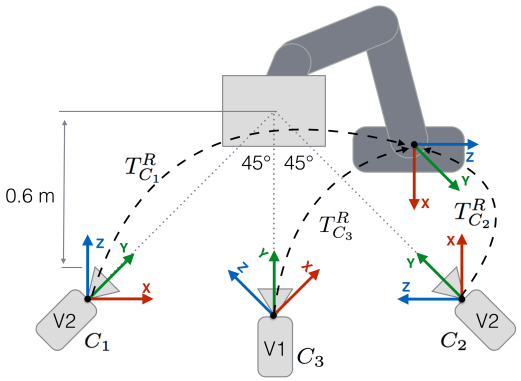


Fig. 1. System Setup with two Kinect V2 depth sensors aimed at the robot end effector at approximately 45° viewpoints and a Kinect V1 sensor placed between them facing the robot. In the system, Eye-to-Hand calibration is represented by the Affine transformation matrix T^R_C, which transforms the coordinate system of each camera to the coordinate system of the robot base, making it common for the whole setup.

Normally, it is sufficient to perform an internal camera parameter calibration only once per sensor unless the lens or sensor itself will be changed or modified. Reliable calibration methods already exist, which are widely used [10] [11] [12] [13].

Eye-to-Hand calibration, on the other hand, is more application specific and crucial for precise environment sensing by the robot or vision guided robot control (visual servoing) [14]. Some work has been successful in calibrating multiple cameras and a robot using a custom-made target object placed in a common field of view for all the sensors in the workspace [15]. Another method calibrated multiple cameras fixed on a rig using structure-of-motion method to estimate relative positions between the cameras [16]. A similar approach was used for calibrating a network of Kinect sensors aimed at robotic inspection of large work-spaces, where sensors are in fixed positions [17]. Robot arm mounted camera, also known as Eye-in-Hand, calibration by moving it to the points of the calibration grid, which is in a fixed position was also proposed [18] [19]. However, most of the presented work is either aimed at the very specific setup or requires a large amount of manual placement of calibration grids, making it time-consuming.

This paper presents a framework to be used for an automatic combined internal camera parameter and Eye-to-Hand calibration by utilizing a robot arm manipulator to actively move around a standard checkerboard calibration grid. The framework is using existing and reliable calibration approaches, but is based on a novel combination of methods to make the calibration process fully automatic and adaptable to as few or as many external 3D cameras as needed. Furthermore, an end-effector to the checkerboard offset is estimated, so a variety of end-effector attachments can be used. It is a time saving and flexible process without requiring any additional equipment for preparing the setup, just a slightly modified A4 size printed checkerboard.

The whole framework is based on the Robot Operating System (ROS) and making use of the modular design and available integration for a large amount of robot and sensor types [20]. Each part of the algorithm is split into a number of separate modules communicating in between each other using pre-defined message formats. The benefits of this approach is the ability to easily modify parts of the process without affecting the rest of processing as well as to include additional processing steps if needed. Furthermore, each framework module can be reused given that the input and output inter-module message format matches.

This allows the actual hardware, robot and 3D cameras, to be interchangeable by simply modifying the configuration file, as long as they have ROS-supported drivers. Only minimal supervision is required during the whole process.

This paper is organized as follows. We present the system setup in Section II. Then, we explain the method in Section III. We provide experimental results in Section IV, followed by relevant conclusions and future work in Section V.

II. SYSTEM SETUP

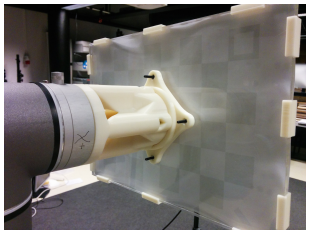
The system setup consists of two main hardware elements: a robot arm manipulator and one or more depth 3D sensors with a visual camera, in our case Kinect sensors.

With the main goal of achieving an environment-aware robot arm manipulator, the robot is thought to be in the center of the setup with sensors observing it from surrounding angles. Positions of the sensors do not need to be fixed, however, in case one of them is being repositioned, the Eye-to-Hand part of the calibration process has to be repeated.

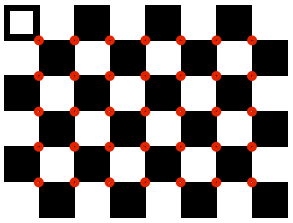
In the described setup, two Kinect V2 depth sensors were used, observing the robot arm end effector from two viewpoints, each angled at approximately 45° and one Kinect V1 facing the robot directly. The setup can be seen in Figure 1. However, the number of sensors is flexible, and only one, or as many as needed can be used as long as sufficient computing power is provided.

A. Calibration Checkerboard

A custom end-effector mount to hold a checkerboard, with an extension to reduce the number of robot self-collisions, was 3D printed and attached to the end-effector, shown in Figure 2(a). The checkerboard contains 7 by 5 squares, each one of 30 mm by 30 mm size, printed on an A4 paper sheet, which is mounted on hard plexiglass surface to prevent any deformation. One of the side squares is modified to be hollow, as shown in Figure 2(b), and is used to identify correct orientation as described in Section III.



(a) A custom end-effector mount with a rigid plexiglass base for holding a checkerboard.



(b) Detected square intersection points are marked in red and a hollow square in the top-left corner, for orientation detection.

Fig. 2. Checkerboard and a custom robot mount.

B. Robot

The robotic manipulator being used is UR5 from Universal Robots with *6 degrees of freedom*, a working radius of *850 mm* and a maximum payload of *5 kg*. The repeatability of the robot movements is *0.1 mm*.

C. Sensors

In our research we include the novel low-cost Kinect V2 sensor [5]. It has been shown to achieve a significantly higher accuracy compared to its predecessor Kinect V1 [13]. Kinect V2 is based on *time-of-flight (ToF)* approach, using a different modulation frequency for each camera, thus allowing multiple ToF cameras to observe the same object without any interference [12]. For comparison reasons, and to demonstrate the flexibility of the system, one Kinect V1 sensor is also included in our setup. Table I summarises technical specifications of Kinect V1 and V2 sensors. Despite both sensors being named Kinect, they are significantly different, requiring separate drivers and, as it was mentioned, are based on different sensing approaches. In general, any 3D camera, with ROS support, can be used with our system.

TABLE I. Kinect V1 and V2 Technical Specifications.

	Kinect V1	**Kinect V2**
Sensor type	Structured Light	Time-of-Flight
RGB Cam Resolution	640x480	1920x1080
IR Cam Resolution	320x240	512x424
Refresh Rate	30 Hz	30 Hz
Depth Range	0.4 to 4.5 *meters*	0.5 to 4.5 *meters*
Field of View Horizontal	57°	70°
Field of View Vertical	43°	60°

D. Software

The whole system software is based on the Robot Operating System (ROS), an open-source meta-operating system running on top of Ubuntu 14.04 [20]. The main advantage of using ROS is its modular design allowing the algorithm to be divided into separate smaller modules performing separate tasks and sharing the results over the network. The workload

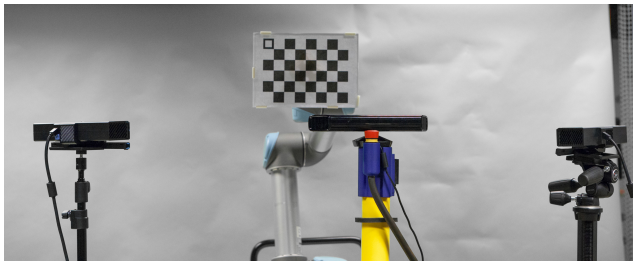


Fig. 3. Picture of the setup. A checkerboard with a hollow square to allow the detection of its orientation is attached to the robot.

in our setup was divided over multiple machines, one for each of the 3D cameras and a central one coordinating all the modules and controlling the robot.

Kinect V2 is not officially supported on Linux, however, open-source drivers including a bridge to ROS were found to function well, including the GPU utilisation to improve the processing speed of large amounts of data produced by sensors [11]. Well tested OpenNI 2 drivers were used to integrate Kinect V1 into the system.

The modular design allows for interchanging any of the modules without the need to make any modifications to the rest of the system. For example, any of the depth sensors can be exchanged to another model, or another robotic manipulator can be used, as long as the inter-modular message format is kept the same. Furthermore, addition of extra depth sensors to the system only requires adding an extra message topic for the coordinating module to listen to.

III. METHOD

Our proposed automatic calibration approach consists of a number of modules working together to achieve the desired accuracy of calibration. The calibration can be divided into two main parts:

1) Sensor internal parameter calibration
2) Eye-to-Hand calibration

We first present the general overview of the system functionality and then go into details of each of the processes.

A. Overview of the Whole System Functionality

The structure of the whole calibration framework is shown in Figure 4. A specific processing is performed by each module and the information between modules is exchanged using custom messages. Instead of having one central unit, each module publishes messages on defined topics to which other modules can subscribe to, resulting in an asynchronous direct peer-to-peer communication. Each message has a time-stamp to allow synchronization and ignoring out-of-date messages. Updating or interchanging modules can be done even at run time as long as the message format is kept identical. Additional sensors can be added in a similar manner, with the new sensor's message topics, which stream IR and RGB images, added to the configuration file, so that it is seen by the rest of the system. It has to be made sure that each camera uses unique message topic names. An overview of the whole calibration process is presented below. Algorithm 1 describes a step-by-step process performed for each camera after the system is launched and 360° initialization movement is performed.

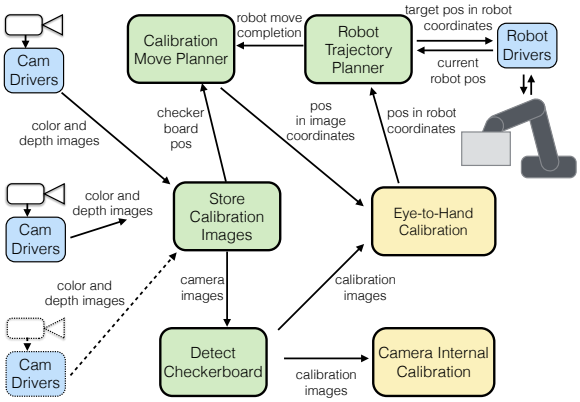


Fig. 4. The whole framework overview including all the modules and the sequence of the processes. Drivers are marked in blue, image analysis and move planning modules are marked in green and actual calibration modules are marked in yellow. A possibility to add additional 3D cameras to the system is represented by the objects in dashed lines.

Algorithm 1 Calibration process for each camera

```
Initial Eye-to-Hand calibration
Tilting motion to define max angles
Estimate the end-effector attachment offset
Generate the robot movement trajectory
loop
  Move the robot to the next position
  Detect checkerboard
  if detected then
    Save images
    Calculate the accumulative Eye-to-Hand calibration
    Apply this calibration
    Recalculate the remaining robot movement trajectory
  end if
end loop
All robot movements are finished
Calculate the internal calibration using saved images
Convert saved images using internal calibration
Calculate the full Eye-to-Hand calibration
```

B. Checkerboard Detection

Existing algorithms included in the OpenCV library were used for checkerboard detection in both color and depth data [21] [7]. Real-time performance is achieved with X and Y coordinates of identified intersection points of squares on the checkerboard, defined as corners, shown in Figure 2(b), and depth value obtained from the depth data. Given the noisy depth data, a normalized value from the surrounding area of 10 pixels over 5 consecutive frames is taken and a median value was calculated to reduce the effects of the sensor noise.

Positions in 3D coordinates of the same checkerboard corners are simultaneously calculated using the robot encoder data, corrected with the estimated offset of the checker-

board mounting. Initially, the assumption is made that the checkerboard center point matches the end-effector center point. Then tilting motions of the end effector in place are performed while observing changed positions of the checkerboard corners. Calculation 3D affine transformation and error minimization between expected corner positions and real ones, provides an accurate offset of the end-effector mount. The end-effector mount has to be rigid. Both the data from 3D cameras and from robot encoders are fully synchronised according to the timestamps of when it was captured to reduce any accuracy issues.

Given the four possible orientations of the checkerboard, the modified corner square of the checkerboard, seen in the top left of the checkerboard in Figure 2(b), is detected using binary thresholding method and the orientation is noted. With the collected data, the corresponding checkerboard corner data points can be matched.

C. Sensor Internal Parameter Calibration

RGB-D cameras are calibrated for internal parameters using the method proposed by Zhang [22] in order to compensate for the following systematic errors:

1) Color camera lens distortion
2) Infrared (IR) camera lens distortion
3) Reprojection error, or color to depth image offset
4) Depth distortion

Other non-systematic and random errors like amplitude-related errors or temperature-related errors are not discussed or analysed in this paper, because standard internal camera parameter calibration procedure does not compensate for them, and they are not crucial in current application [5] [6].

D. Eye-to-Hand Calibration

Using the corresponding 3D corner points of the calibration checkerboard, a 3D Affine transformation matrix between the 3D camera and the robot end effector is estimated [7]. With some likelihood of imprecise detection of checkerboard corners, the outlier detection based on Random Sample Consensus (RANSAC) method is being used on the inputs [23]. The outcome of the estimator is a 3x4 Affine transformation matrix seen in Equation 1, where R is a $3x3$ rotation matrix and t is $3x1$ translation vector.

$$T_{C_3}^{R} = \begin{Bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{Bmatrix} \quad (1)$$

Using the calculated transformation matrix, the 3D points detected in 3D camera color image and depth data can be transformed from the camera coordinate system to the robot's base coordinate system.

E. Robot Motion Planning

Robot arm control in Cartesian coordinates is used in the project, given the relatively simple movements, as well as limited workspace. Multiple motion planning algorithms included in the *MoveIt!* framework [24] were tested. The RRT-connect approach [25], based on the Rapidly exploring random tree, was found to be suitable for the task. We used an implementation from the from the Open Motion Planning Library (OMPL) [26].

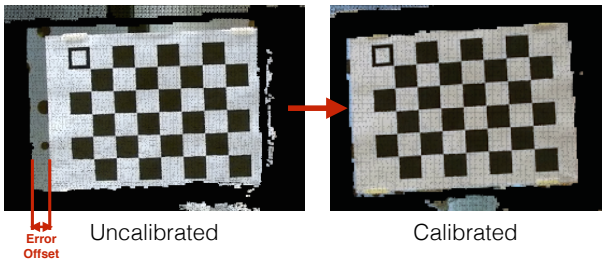


Fig. 5. Reprojection error shown in the color image and depth point cloud overlay. The offset in the left image is caused by imprecisely defined relative positions between the color and infrared cameras in the 3D camera. Internal camera calibration compensates for this error. The result is seen in the image on the right side, where the offset is reduced.

In order to achieve a high-quality 3D camera internal calibration, the samples should include the checkerboard positioned in the majority of the camera's field of view and for least at two distances. Furthermore, tilting the checkerboard at different angles in relation to the camera increases the calibration accuracy [11]. Knowing the calibration pattern parameters, tilting motion allows for more accurate mapping from 3D world coordinates to the 2D image sensor coordinates, based on the projection lines of the known calibration pattern [22].

In order to simplify the internal calibration, it was decided to collect all the data at once and then the calibration using the whole dataset was calculated. However, this meant that lens distortion was still present during the data collection. Furthermore, a reprojection error occurs, which is an offset between the color image and depth data, shown in Figure 5. These issues caused the Eye-to-Hand transformation to be imprecise, especially for points closer to the edge of the camera image, where lens distortion is more significant.

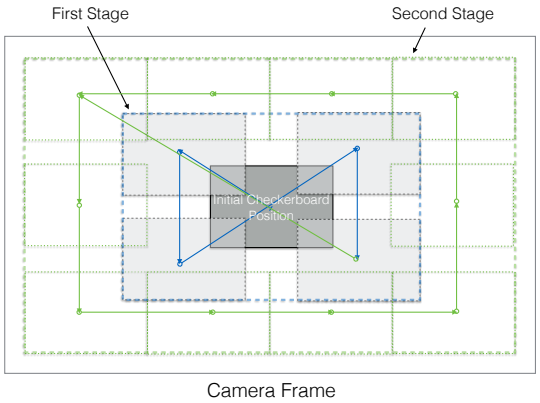


Fig. 6. Robot movement trajectory as seen in the 3D camera image. It is split into multiple stages by the positions calculated at increasing distance from the center point of the image. Movements are done stage-by-stage, while improving the Eye-to-Hand transformation accuracy at each step. This figure shows just a two stage example.

A robot movement trajectory was chosen with small offsets from the starting point. It can be split into multiple stages by the positions calculated at increasing distance from the center point of the image. The number of stages depend

on the overlap level of the checkerboard positions in the camera image, size of the checkerboard, reach-ability of the robot arm as well as the size of the area covered by the camera. At each new position, the detected checkerboard intersection points are accumulated and Eye-to-Hand calibration was recalculated to continually improve the accuracy. The example two-stage robot movement trajectory is shown in Figure 6. However, this data is not considered for the final Eye-to-Hand calibration, because the 3D camera sensor itself is still not calibrated at this point.

This approach has shown to reduce the robot movement error and by the time positions close to the image edges are chosen, the transformation is accurate enough not to exit the camera's field of view, where checkerboard corners cannot be detected anymore.

During the calibration movements, the checkerboard is tilted by defining changing roll, pitch and yaw of the end effector. During the initialisation of the calibration, a checkerboard is turned to face the 3D sensor directly and then tilted to each direction at 5 degree increments, both positive and negative rotation direction, while trying to detect the checkerboard. Once the detection fails, the tilting is backtracked and the angle is saved as a maximum allowed tilting. The same process is performed for roll, pitch and yaw to positive and negative angle limits. Roll angle is limited to $\pm 45^{\circ}$. This angle can vary significantly depending on the type of 3D camera as well as lighting conditions. Reflections caused by the room lights can affect the checkerboard detection.

At each position, images and detected 3D corner positions in color and infrared camera images are saved at each pose. If the desired point is outside the robot workspace, it is automatically identified by the planning algorithm and skipped. Given the positions are reachable, the same trajectory is performed with a 20% higher depth offset away from the camera in order to have data at different distances from the sensor calibrated, while the checkerboard still appears large enough in the image to be detected.

Once the planned movement trajectory is completed, internal 3D camera calibration is performed using the collected data.

F. Repeated Eye-to-Hand Calibration

After the internal calibration of each 3D camera, the accuracy of the Eye-to-Hand calibration is not precise because of compensated lens distortion and adjustments to reduce the reprojection error. The simplified move sequence, without the tilting, is repeated with the robot moving to previously visited positions by reusing the same coordinates and just Eye-to-Hand calibration recalculated. Once this process is finished, the sensor in the system is fully calibrated.

G. Checkerboard Observation

As mentioned previously, a flexible number of 3D cameras can be calibrated with the system. Inclusion of the additional sensor into the system is done by defining a configuration file containing the topic names the camera is publishing on. While one 3D camera is being calibrated, any other sensors included in the system are passively observing the robot and running the simplified checkerboard detection algorithm. If the checkerboard is detected, the pose of the checkerboard and the pose of the robot, which is being streamed on the network by the robot controlling node, are recorded. In any subsequent checkerboard detection instances, the position is compared to the position of the previous detection, and if the current one is closer to the center of the color image, the poses are updated in order to have a more reliable starting position. Once the current calibration of a 3D camera is completed, the request is sent to the robot to move to the detected position and start the calibration procedure for the other sensor.

IV. Experiments and Results

The presented calibration process was successfully performed provided that the checkerboard was detected by the 3D camera to be calibrated. Initially, the robot was placed in an upside-down "L" shaped joint configuration and turned 360° to increase the chances of the checkerboard being detected by the 3D sensors. However, this movement has to be supervised by a human operator to avoid hitting any obstacles. In other cases, the robot was repositioned manually to make sure that the checkerboard was within the field of view of the camera.

Given a close to autonomous operation of our framework, we conducted experiments to analyze the number of checkerboard positions recorded versus the achieved calibration accuracy. As the process for the internal sensor calibration is identical for each of the 3D cameras, for easier comparison, the results from one Kinect V2 camera is presented in the experiments section. Meanwhile, the Eye-to-Hand calibration results have been acquired using the setup described in Section II. Results are divided into two sections according to the calibration type, each one requiring an independent set of robot moves and data collection:

1) Internal camera calibration
2) Eye-to-Hand calibration

A. Internal Camera Calibration

The first iteration of movements was made in order to calibrate the 3D camera internally, using the robot trajectory explained in Figure 6. Because the field of view of the color camera and the infrared camera in the sensor are different, the checkerboard was not always visible or successfully detected in both cameras at the same time. This explains the varying number of detections in each of the sensor's cameras, as well as simultaneously in both, which we refer to as *combined*. There were 9 experiments conducted in total. Experiments 1-2 had large overlap in checkerboard positions and tilting, experiments 3-6 had no overlap anymore and experiments 7-9, no more tilting. Experiment data is summarized in Table II.

Figure 7 shows the calibration results by analyzing the average error in pixels of each of the sensor's cameras and the reprojection error for each of the experiments. Errors were calculated using the known geometry and size of the

TABLE II. Experiment data for internal Kinect V2 sensor calibration.

Exp #	Color Frames	IR Frames	Combined Frames	Overlap	Tilting	Time (sec)
Exp 1	234	215	158	Yes	Yes	613
Exp 2	120	109	81	Yes	Yes	338
Exp 3	78	72	55	No	Yes	218
Exp 4	57	54	45	No	Yes	176
Exp 5	44	41	33	No	Yes	142
Exp 6	39	35	26	No	Yes	128
Exp 7	15	14	14	No	No	57
Exp 8	10	9	7	No	No	45
Exp 9	5	5	5	No	No	36

checkerboard and comparing the calibrated sensor estimation of the checkerboard dimensions according to the square intersection points to the known geometry. The higher the error, the lower the calibration accuracy.

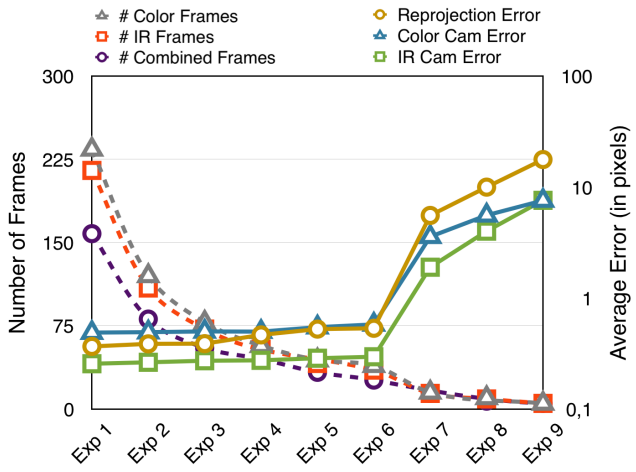


Fig. 7. Calibration accuracy results by showing the errors of internal 3D camera calibration. Color camera, IR errors define averages of each sensor's cameras. Reprojection error defines the average error of the offset in the images between the color image and the depth information, seen in Figure 5. It has to be noted that right Y axis for error rates is in log scale.

B. Eye-to-Hand Calibration

TABLE III. Experiment data for Eye-to-Hand calibration.

Exp #	Frames Cam 1	Frames Cam 2	Frames Cam 3	Overlap	Time (sec)
Exp 1	82	80	71	Yes	470
Exp 2	44	45	39	Yes	243
Exp 3	14	14	11	No	115
Exp 4	9	10	8	No	85
Exp 5	5	6	5	No	60

The second iteration of moves were performed for Eye-to-Hand calibration, while using the most accurate internal 3D camera calibration mentioned previously. In this part, tilting was not performed and the calibration checkerboard was kept at a constant angle, parallel to the each camera's image plane. 5 experiments using 3 cameras were conducted in total, each using a different number of frames, as seen in Table III. Experiment 1 had a large overlap in checkerboard positions, in experiment 2 there was a small overlap, while in the rest there was no overlap and even some gaps between the positions. Cam 1 and 2 were Kinect V2 sensors, while Cam 3 was Kinect V1. Time was measured from the start to the final calibration result of all three cameras.

C. Result Analysis

For the internal calibration, it can be seen that in the first 6 experiments, even with a significantly lower number of frames used, the error in all of the sensor's cameras did not increase much. However, experiments 7 to 9, where the calibration checkerboard was present only in the part of the camera's field of view and was not tilted, show a significant increase in errors. It can be concluded that the most important part to achieve good internal calibration accuracy is to cover the field of view of the camera and perform tilting, but overlapping same areas with the checkerboard is not mandatory.

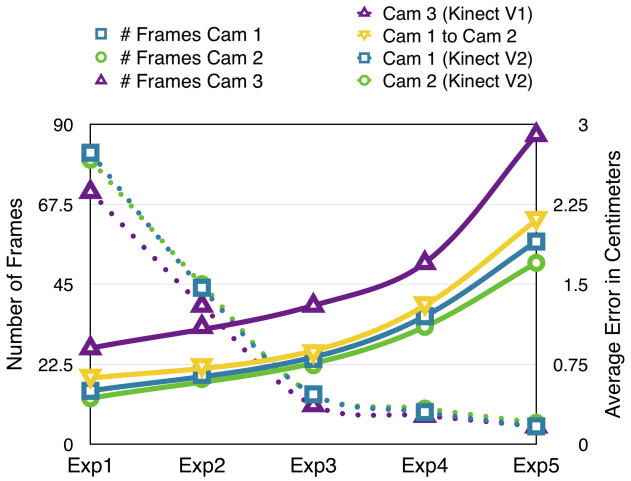


Fig. 8. Eye-to-Hand calibration accuracy results. Overall position error (in cm) as well as each axis separately are shown by comparing the actual robot position versus the predicted robot position from the 3D camera sensor. Dotted lines indicate the number of frames used in each experiment.

Figure 8 shows that the average error rate of Eye-to-Hand calibration has inverse correlation to the number of frames used. The larger area of the camera's field of view is used, the more accurate calibration is achieved. Looking at the overall average error, experiment 3 seems to be the most optimal choice for all three cameras when considering the number of frames used and accuracy achieved.

As expected, the older Kinect V1 was significantly less accurate compared to Kinect V2, mainly due to lower resolution RGB and IR sensors in the camera. The average error between camera 1 and camera 2, both being Kinect V2, was almost the same as average errors of each sensor to the robot. Therefore, calibration of each 3D camera to the robot is enough for joining the point clouds of two cameras, and no additional calibration is necessary.

It was also noticed in all five experiments that the Z-axis has on average 20% larger error compared to both X and Y-axis. It is likely to be caused by the noisy depth data from 3D camera, which should be compensated using more specific methods. On the other hand, it proves that position estimation in X and Y-axes has even lower error than our indicated overall error of the calibration.

V. CONCLUSION AND FUTURE WORK

A simple and flexible calibration method for systems containing a robot and one or more 3D cameras was presented. It is based on the robot moving a standard calibration checkerboard and being guided by the information sent from each of the cameras to cover the largest possible area in the field of view, to ensure an accurate calibration.

A full calibration, including a sensor internal calibration together with an Eye-to-Hand calibration can be done, or just the second part separately, given that the sensor is already calibrated internally. Modular design ensures that sensors can be added or removed easily, as well as hardware components interchanged without any modifications to the algorithm.

According to experiment results, achieving good calibration requires the robot to cover the majority part of the field of view of the 3D camera to achieve a good accuracy. Using our system, a good accuracy calibration of one 3D sensor taken just out of the box, can be achieved in just a few minutes with minimal supervision by the operator. This makes the system integration and reconfiguration significantly faster compared to standard manual method, while keeping the flexibility of varying configurations.

An example application the calibration process was aimed at the environment-aware collaborative robot arm, where people or other moving objects can freely and safely operate in the workspace of the robot without a risk of collision. However, for a more precise operation where sub-centimeter accuracy is necessary, a different, more up-close, setup would be needed as well as a checkerboard containing a finer structure.

The framework will be further tested with a variety of physical setups and different 3D cameras and multiple robot arm types. We plan to open source the code, making it accessible to researchers allowing further testing and development.

Algorithm improvements will include a simultaneous calibration of multiple cameras provided that the calibration checkerboard is within the field of view. Furthermore, if only part of the field of view of the camera will be used in the operation, it could be defined by the user and instead of calibrating the whole image area, only the area of interest would be used.

ACKNOWLEDGMENT

This work is partially supported by The Research Council of Norway as a part of the Engineering Predictability with Embodied Cognition (EPEC) project, under grant agreement 240862

REFERENCES

[1] I. Bonev, "Should We Fence the Arms of Universal Robots?" http://coro.etsmtl.ca/blog/?p=299, ETS, 2014, accessed September 7, 2015.

[2] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 338–345.

[3] P. Rakprayoon, M. Ruchanurucks, and A. Coundoul, "Kinect-based obstacle detection for manipulator," in *System Integration (SII), 2011 IEEE/SICE International Symposium on*. IEEE, 2011, pp. 68–73.

[4] C. Fitzgerald, "Developing baxter," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[5] P. Fankhauser, M. Bloesch, D. Rodriguez, , R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *IEEE International Conference on Advanced Robotics (ICAR) (submitted)*, 2015.

[6] J. Smisek, M. Jancosek, and T. Pajdla, "3D with Kinect," in *Consumer Depth Cameras for Computer Vision*. Springer, 2013, pp. 3–25.

[7] OpenCV, "Camera calibration and 3d reconstruction," *Online Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html*, 2015.

[8] S. Ma and Z. Hu, "Hand-eye calibration," in *Computer Vision*. Springer, 2014, pp. 355–358.

[9] R. Horaud and F. Dornaika, "Hand-eye calibration," *The international journal of robotics research*, vol. 14, no. 3, pp. 195–210, 1995.

[10] R. Y. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 4, pp. 323–344, 1987.

[11] T. Wiedemeyer, "IAI Kinect2," https://github.com/code-iai/iai_kinect2, Institute for Artificial Intelligence, University Bremen, 2014 – 2015, accessed June 12, 2015.

[12] S. Foix, G. Alenya, and C. Torras, "Lock-in time-of-flight (tof) cameras: A survey," *Sensors Journal, IEEE*, vol. 11, no. 9, pp. 1917–1926, 2011.

[13] C. Amon, F. Fuhrmann, and F. Graf, "Evaluation of the spatial resolution accuracy of the face tracking system for kinect for windows v1 and v2," in *Proceedings of the 6th Congress of the Alps Adria Acoustics Association*, 2014.

[14] V. Lippiello, B. Siciliano, and L. Villani, "Eye-in-hand/eye-to-hand multi-camera visual servoing," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*. IEEE, 2005, pp. 5354–5359.

[15] T. Heikkilä, M. Sallinen, T. Matsushita, and F. Tomita, "Flexible hand-eye calibration for multi-camera systems," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2000, pp. 2292–2297.

[16] S. Esquivel, F. Woelk, and R. Koch, "Calibration of a multi-camera rig from non-overlapping views," in *Pattern Recognition*. Springer, 2007, pp. 82–91.

[17] R. Macknojia, A. Chávez-Aragón, P. Payeur, and R. Laganiere, "Calibration of a network of kinect sensors for robotic inspection over a large workspace," in *Robot Vision (WORV), 2013 IEEE Workshop on*. IEEE, 2013, pp. 184–190.

[18] H. Zhuang, K. Wang, and Z. S. Roth, "Simultaneous calibration of a robot and a hand-mounted camera," *Robotics and Automation, IEEE Transactions on*, vol. 11, no. 5, pp. 649–660, 1995.

[19] F. Dornaika and R. Horaud, "Simultaneous robot-world and hand-eye calibration," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 4, pp. 617–622, 1998.

[20] 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 *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[21] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[22] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[23] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[24] I. A. Sucan and S. Chitta, "MoveIt!" *Online Available: http://moveit.ros.org*, 2013.

[25] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[26] I. Sucan, M. Moll, L. E. Kavraki, *et al.*, "The open motion planning library," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 72–82, 2012.

II

PAPER II

Multi 3D camera mapping for predictive and reflexive robot manipulator trajectory estimation

J. Miseikis, K. Glette, O. J. Elle and J. Torresen

2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, 2016, pp. 1-8.

Multi 3D Camera Mapping for Predictive and Reflexive Robot Manipulator Trajectory Estimation

Justinas Mišeikis*, Kyrre Glette*, Ole Jakob Elle*† and Jim Torresen*
*Department of Informatics, University of Oslo, Oslo, Norway
Email: {justinm, kyrrehg, oleje, jimtoer}@ifi.uio.no
†The Intervention Centre, Oslo University Hospital, Oslo, Norway

***Abstract*—With advancing technologies, robotic manipulators and visual environment sensors are becoming cheaper and more widespread. However, robot control can be still a limiting factor for better adaptation of these technologies. Robotic manipulators are performing very well in structured workspaces, but do not adapt well to unexpected changes, like people entering the workspace. We present a method combining 3D Camera based workspace mapping, and a predictive and reflexive robot manipulator trajectory estimation to allow more efficient and safer operation in dynamic workspaces. In experiments on a real UR5 robot our method has proven to provide shorter and smoother trajectories compared to a reactive trajectory planner in the same conditions. Furthermore, the robot has successfully avoided any contact by initialising the reflexive movement even when an obstacle got unexpectedly close to the robot. The main goal of our work is to make the operation more flexible in unstructured dynamic workspaces and not just avoid obstacles, but also adapt when performing collaborative tasks with humans in the near future.**

I. INTRODUCTION

In many practical applications, industrial robot manipulators are still working "blind" with hard-coded trajectories. This results in the workspace for robots and humans being strictly divided in order to avoid any accidents, which, unfortunately, sometimes still occur. It is often more common to have collision detection systems, which do not always work as expected, rather than collision prevention methods [1]. However, *environment-aware robots* [2] [3] are becoming more common, both developed in research and by robot manufacturers themselves (e.g. Baxter by Rethink Robotics) [4].

The theme of shared workspace has been researched for many years, however it is still a highly relevant research topic today [5] [6]. Workspace sharing can be classified as *robot-robot* and *human-robot* systems for task sharing, collaborative or supportive tasks. Normally, in *robot-robot* sharing, controllers of all involved robot systems have direct communication and can coordinate moves easier by knowing the planned trajectories for all the manipulators. We will be focusing on *human-robot* shared workspaces, where sensors are used to observe the environment and adapt the manipulator behaviour according to movements in the workspace, normally caused by human motion.

There have been a number of systems proposed addressing the issue of robot trajectory planning in shared workspaces. One system runs a genetic algorithm using fuzzy logic and defines all obstacles as static while the new trajectory is found [7]. However, it is not suitable for obstacles moving at higher velocities. Another work provides an analysis of non-verbal cues given by humans and robots, and shows that movement understanding plays an important role in the usability of a system and the *human-robot interaction (HRI)* [8].

Some of the systems for *human-robot* interaction assign the robot arm as a light manipulator, thus, reducing a possible collision force and then using inertia reduction, passivity and parametric path planning [9]. However, this method leads to light collisions, which, ideally, should be avoided.

With camera systems, especially 3D cameras, becoming more affordable, obstacle detection in the robot workspace becomes easier. Sophisticated methods based on robot manipulator modeling and obstacle motion estimation allow a rapid recalculation of robot trajectories to avoid collision with moving obstacles [2]. Another system uses multiple Kinect cameras observing the same workspace from different viewing points to avoid collisions, but no definite planning approach was proposed [10]. Furthermore, a system was proposed using the historical data of obstacle positions in the workspace of the robot. It avoids the areas which are commonly occluded by a human and plan movement trajectories around them [11]. However, there is a high risk of a collision if the obstacle appears right in front of the manipulator and is not modeled yet. Such situations can occur, when the obstacle was not seen by the camera before it got too close to the robot, for example due to an occlusion or a blind spot of the camera.

Most of the presented approaches rely on one method, commonly a reactive trajectory re-planning to an unexpected obstacle. We propose a method combining a two layered trajectory planner for a manipulator working in a shared *human-robot* workspace. Multiple 3D cameras are used to observe the workspace from different viewpoints to reduce the chance of occlusions. At the same time, a danger map of the workspace is created indicating the areas which are commonly entered by an obstacle, e.g. person's arm. The system contains two behaviour models. *Reflexive behaviour* immediately reacts to unexpected obstacles appearing close to the robot. *Predictive behaviour* uses the danger map information to predict the probability of an obstacle entering areas of the manipulator workspace and avoids it in advance. The most optimum trajectories are estimated considering the probability of a collision with the obstacle as well as the distance traveled by the end effector of the robot. Collision prevention is done not just for the end

effector, but for the whole body of the robot.

One of the possible applications of our proposed method is a surgery assistive robot in an operating theater, where simple tasks like holding a probe or handling surgical tools will be automated. This requires a guaranteed safety with no unexpected impact with a patient or staff around, as well as surrounding equipment. In some cases, there might be multiple robots working in collaboration, for example a robot with a C-arm mounted fluoroscopy scanner working in parallel. Direct communication and motion planning are not always possible, so our proposed method provides an appropriate alternative solution.

Furthermore, with classification of obstacle types, *reflexive behaviour* model can be adapted for collaborative tasks, where a person can hand over objects to the robot or use it for support.

This paper is organized as follows. We present the system setup in Section II. Then, we explain the proposed method in Section III. We provide experimental results in Section IV, followed by relevant conclusions and future work in Section V.

II. SYSTEM SETUP

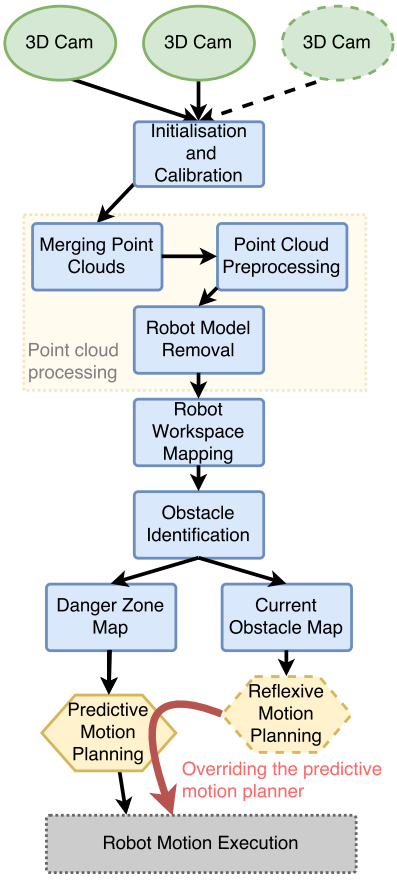


Fig. 1. Overview of our proposed method. Green ovals represent sensing part, blue rectangles - processing part, yellow hexagons - motion planning and gray rectangle (dashed borders) - motion execution. Reflexive motion planning marked as yellow hexagon (dashed borders) overrides the predictive motion planning when an unexpected obstacle gets close to the robot. There can be a variable number of 3D cameras included in the system.

A. Hardware

The robotic manipulator being used is UR5 from Universal Robots with *6 degrees of freedom*, a working radius of *850 mm* and a maximum payload of *5 kg*. The repeatability of the robot movements is *0.1 mm*.

In our research we include a low-cost Kinect V2 sensor [12]. It has been shown to achieve a significantly higher accuracy compared to its predecessor Kinect V1 [13]. Kinect V2 uses a *time-of-flight (ToF)* approach, using a different modulation frequency for each camera, thus, allowing multiple ToF cameras to observe the same object without any interference [14]. For short-range sensing, an Intel F200 3D camera was mounted on the end-effector to detect any obstacles, which are in close proximity of the end effector [15]. Also one Kinect V1 sensor is included in our setup. In general, any 3D camera, with ROS support, can be used with the system.

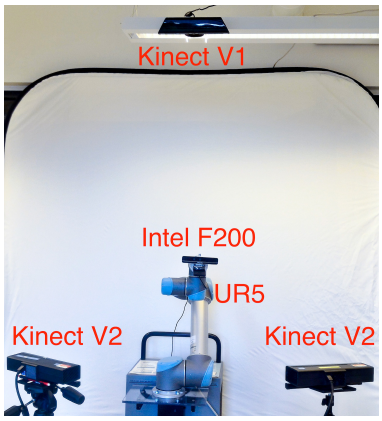


Fig. 2. Our system setup with overhead Kinect V1, two Kinect V2 cameras placed at different angles observing the front of the robot and an Intel F200 camera mounted on the end effector of the UR5.

B. Software

The system software runs on the Robot Operating System (ROS) running on Ubuntu 14.04 [16]. The main advantage of using ROS is its modular design allowing the algorithm to be divided into separate smaller modules performing separate tasks and sharing the results over the network. The workload in our setup was divided over multiple machines.

Kinect V2 is not officially supported on Ubuntu, however, open-source drivers including a bridge to ROS were found to function well, including the GPU utilisation to improve the processing speed of the large amounts of data produced by the sensors [17]. Well tested OpenNI 2 drivers were used to integrate Kinect V1 and Intel F200 into the system.

III. METHOD

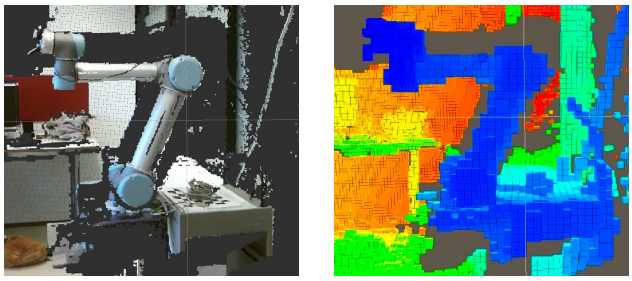
A. System Overview

Our system contains a number of processes working both in series and in parallel as seen in Fig. 1. Below we present each part of the algorithm in more detail.

B. Calibrating 3D Cameras to the Robot

The first step of the system setup is to place the 3D cameras around the robot. The goal is to observe the complete robot workspace and to avoid any occlusions. In our case, two Kinect V2 cameras facing the robot were placed, angled at 45° relative to the robot base, one Kinect V1 overlooking the system from the top and the Intel F200 camera mounted on the end effector of the robot, as shown in Fig. 2. However, many different combinations can be used and selection should be made depending on the application.

With fixed camera positions the Eye-To-Hand calibration can be performed to map all the 3D camera coordinate systems to match the robot base coordinate system [18] [19]. This can be done automatically by placing a calibration board on the robot's end effector and using the proposed automatic calibration procedure [20]. It uses the estimated checkerboard position and robot joint encoder information to guide the robot movements and cover the field-of-view (FoV) of each of the cameras as much as possible for an accurate Eye-to-Hand calibration.



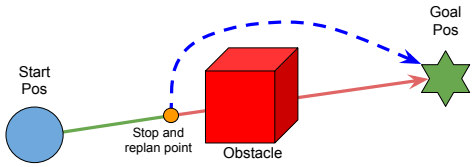
(a) Point cloud (b) Octomap representing the input point cloud

Fig. 3. Octomap created from the Kinect V2 point cloud data of the robot scene. Colorscheme represents the distance of objects from the 3D camera.

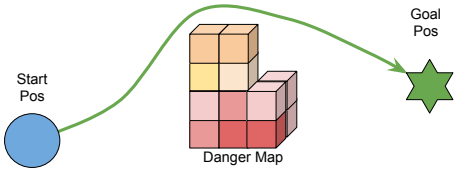
The robot automatically performs a number of moves until a precise calibration is achieved. Using the newly calculated transformation matrix describing the positions of the 3D cameras relative to the robot, all the point clouds can be mapped onto a common coordinate frame originating at the robot base.

C. Merging Point Cloud Data

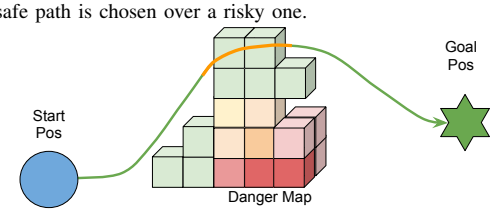
In order to map the whole workspace of the robot, point clouds from each of the 3D cameras have to be merged. The calibration provides a good estimation of the transformation matrices for accurate merging, but additionally, an Iterative Closest Point (ICP) method is used for fine alignment of all the point clouds [21]. The process is performed for each of the cameras. Once the precise transformation matrices have been calculated using the ICP method, they are applied for transformations of all the incoming point clouds. Camera calibration and the ICP method do not need to be repeated unless the cameras or the robot base are moved in relation of each other.



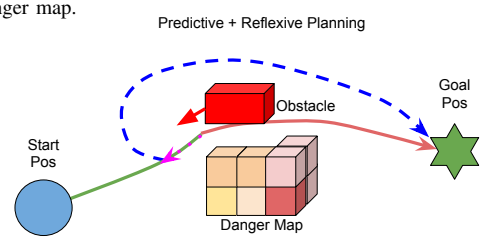
(a) Reactive Behaviour Planning: When the obstacle is present, the robot stops and recalculates the path. Occupied workspace is never crossed.



(b) Our method: the trajectory fully avoids, but gets close to the medium risk area in the danger map. If detour is not large, a safe path is chosen over a risky one.



(c) Our method: the trajectory crosses the low risk area in the danger map.



(d) Our method: Predictive and Reflexive Planning. The obstacle gets very close to the robot resulting in the reflexive behaviour being initialised and a new alternative trajectory calculated.

Fig. 4. Comparison of our proposed joint predictive and reflexive trajectory planning versus a traditional reactive trajectory planning. In danger maps, the scale of risk is from the lowest risk (light green) to the highest risk (red).

D. Point Cloud Pre-processing

In order to increase the processing speed and filter out unwanted noise, a number of pre-processing steps are performed on the input data from each of the 3D cameras.

The first step is to filter out any points in the point cloud data that are too far away from the robot workspace. Knowing that the workspace radius is $850mm$ from the base of the robot, any points that are further than $1500mm$ from the robot base are removed as they are not important in our application. This significantly downsizes the point cloud.

Then, any outliers in the point cloud are removed using a Statistical Outlier Removal algorithm [22]. After noisy data has been removed, point clouds can be simplified by down-sampling using a voxel grid filter, which normally reduces the number of points with a minimal loss of information. The

voxel grid filter performs a smart down-sampling by sub-diving the space containing point cloud data into a set of volumetric pixels (voxels) and all the points inside each voxel are approximated with the coordinates of their centroid.

E. Removing the Robot Model

After merging the point clouds, Eye-To-Hand calibrations together with the precise model of the robot arm and its current configuration in space are used to remove the underlying points of the 3D robotic arm model. This step is necessary to avoid false positives on self-collision, as some parts of the robot, seen by the 3D cameras would be interpreted as an obstacle.

It is done by fitting simple shapes, in this case cylinder models on known robot links by taking current angles of all the joint encoders. Cylindrical models are expanded to be $5mm$ larger than the actual robot links to compensate for any noisy point cloud measurements. Once the model is fitted, any points lying inside the cylinder models are removed under the assumption that the robot itself is represented by these measurements. In order to reduce the computational costs, the shape fitting process is re-done only when the robot moves from the previously fixed position.

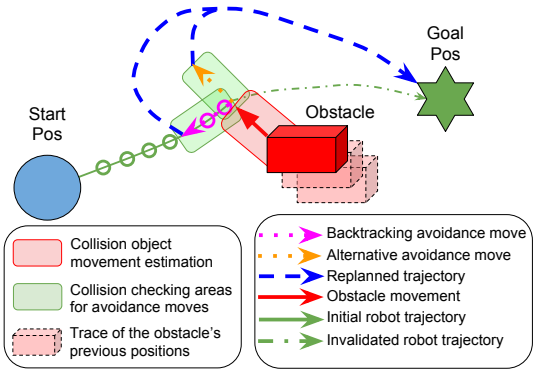


Fig. 5. Reflexive Behaviour. The planned robot trajectory (green solid line) is blocked by a moving obstacle (marked in red). The first option is to backtrack on the executed trajectory(pink dotted line) until the collision risk is over. If backtracking still results in a collision, the second option is to use the alternative avoidance move and move to the direction opposite from where the obstacle is approaching (dotted orange line). In the meantime, an alternative collision-free trajectory to the goal position is calculated and executed (blue dashed line).

F. Mapping

An octomap was chosen as an efficient 3D mapping method of the robot workspace. It is combined of octrees, which are hierarchical data structures for spatial subdivision in 3D. Each node in the octree represents the space contained in a cubic volume, called a voxel. This volume is recursively subdivided into eight sub-volumes until a given minimum voxel size is reached. The minimum voxel size determines the resolution of the octree. The resolution can be dynamically adjusted, both for the whole map, or just parts of it, as each octree branch can be sub-divided into smaller parts [23]. This approach enables us to use a simple structure to represent occupied, free and unknown areas in the map. The conversion from the point cloud into an octomap can be seen in Fig. 3.

Octomaps can be either binary or full. In binary octomaps, each of their voxels have a binary value, where 1 stands for occupied and 0 for free space. They are suitable for immediate reactions because of the quick processing. While in full octomaps, float values between 1 and 0 are used to describe the probability of voxels being occupied or free, and probabilistic functions can be used to adjust them.

G. Danger Area Identification

Any obstacle (e.g. a person entering the workspace of the robot) visible to 3D cameras at the time of the observation is recorded in the octomap as an occupied voxel. As long as the obstacle stays there, the respective voxels will remain occupied. However, when the obstacle is not present anymore in a previously occupied voxel, we do not want to mark it as free immediately. Instead, we introduce a cost function to produce a slow decay, which represents the probability of how risky it is for the robot to enter the area.

1) Cost function: The time dependent cost decay function, shown in Eq. (1), is based on an inverse logarithmic decay to provide a slow decrease at first with an increasing decay the longer the area was not occupied anymore. When the voxel is occupied, its value $C_{voxel,t}$ is reset back to 0.999.

$$C_{voxel,t} = C_{voxel,t-1} + ln(C_{voxel,t-1}) * (\Delta t * \alpha) \quad (1)$$

Parameter α is used to adjust the decay speed and Δt defines the time difference between two calculations, normally determined by the rate of incoming data frames from the camera. The cost function ensures that the areas in the workspace where obstacles are commonly present and their presence is recurring will be mapped as risky to enter for the robot, and it will attempt to find alternative trajectories through the safe areas to reach the next goal position.

H. Robot Motion Planning

Robot motion planning is based on a two-layered structure: *reflexive* and *predictive behaviour*.

1) Reflexive Behavior: For any immediate danger, a *reflexive behaviour* model is used. Inspired by human behaviour of how we immediately move our hand away from anything that is sharp or burning hot, and only then look at the object and think what to do next. Similarly to this, the robot uses the simplified binary octomap consisting only of currently observed obstacles. If any obstacle is categorised as an immediate danger, the reflexive movement is performed. Last couple of already passed waypoints of the current trajectory are taken as a new goal position, and if the path is free, the movement is immediately executed. Otherwise, if moving back down the previously executed trajectory still results in a collision, an alternative avoidance move is initialised. The movement vector is calculated by taking a vector from the end effector of the robot to the closest point of the obstacle and inverting the direction. The *reflexive behaviour* model is explained in Fig.

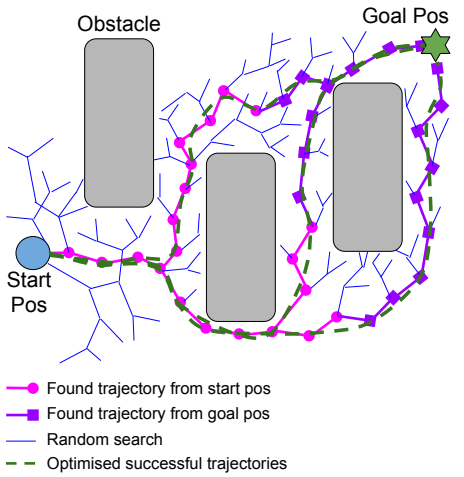


Fig. 6. Visualisation of a two dimensional RRT-connect trajectory planning algorithm. The method is based on growing Rapidly-exploring Random Trees (marked in thin blue) from the start and goal positions until a connected path is found (marked in pink and purple). Once one or more successful trajectories are found, they are optimised and smoothed (green dashed lines) before they are executed on the robot.

5. The risk of collision is determined by Eq. (2), inspired by the braking distance calculations [24]:

$$D_{\text{risk}}(v, D_{\text{eucl}}) = \frac{|v^2|}{\gamma D_{\text{eucl}}} \tag{2}$$

γ is a user set parameter, D_{eucl} is the Euclidean distance between the obstacle and the robot, v is the velocity vector of the obstacle movement. The safe distance is determined by a threshold T, which is normally set to 1 and instead the parameter γ is adjusted. If D_{risk} exceeds the threshold T, a reflexive motion planning overrides the predictive one.

$$behaviour = \begin{cases} reflexive & \text{if } |D_{\text{risk}}| \geq T \\ predictive & \text{otherwise} \end{cases} \tag{3}$$

2) Predictive Behavior: Independently from the previously described *reflexive behaviour*, a predictive re-planning continuously runs to find the safest and most optimum path to the next goal position. The full octomap, including the calculated danger areas is used for prediction. Each octomap voxel contains the cost (danger) value described in Eq. (1). The motion planner punishes the trajectories, which place any part of the robot in the risky areas (voxels containing non-zero cost). The measure being used for the trajectory evaluation is the accumulated distance through each of the octomap voxels and risk levels added as shown in Eq. (4). C_{traj} is the total cost of the trajectory, D_{traj} is the Euclidean distance through the octomap voxel and C_{voxel} is from Eq. (1).

$$C_{traj} = \sum_{voxel} (D_{traj} + D_{traj} * C_{voxel}) \tag{4}$$

This way, a longer trajectory through fully safe areas might be preferred rather than a short and risky one. It has to be noted, that any trajectory execution of the *predictive planner* can be over-ridden by an *reflexive behaviour* planner whenever a high risk obstacle is present. These two planners work in parallel with the reflexive one having higher priority.

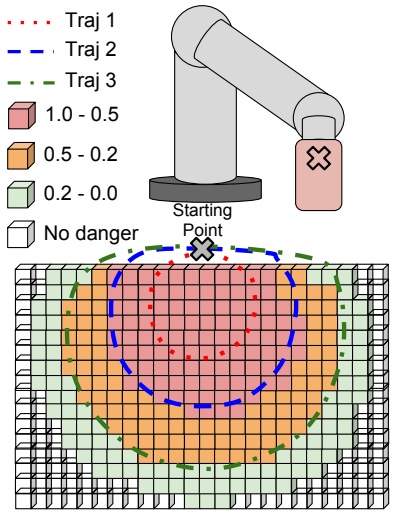
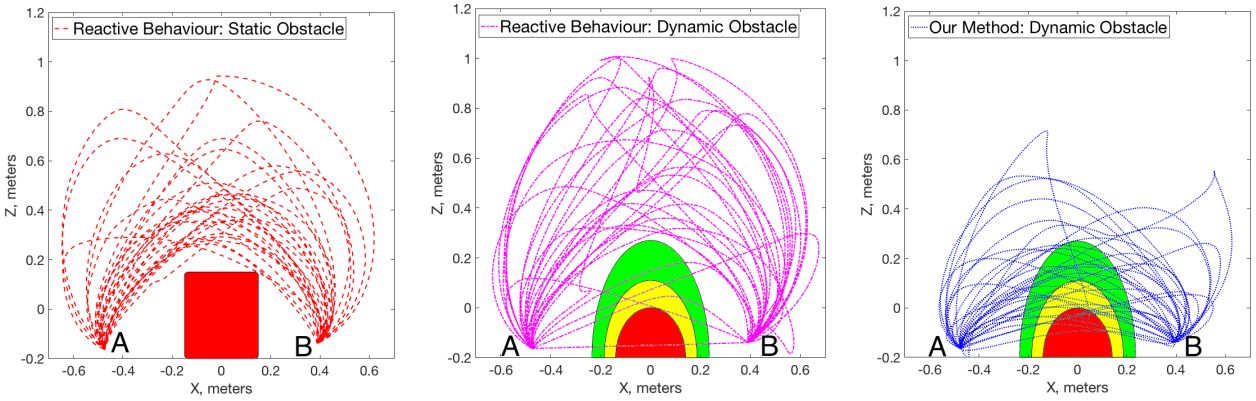


Fig. 7. Danger map creation test. An object acting as the obstacle was mounted on the end effector of the robot. Then the three indicated trajectories were repeatedly executed in the workspace one after another while the danger map was built up and updated. The average danger cost C_{voxel} of each voxel over the period of the whole experiment is shown with colors red, orange and green indicating high, average and low risk accordingly. Transparent cubes show voxels with zero cost, representing areas without danger.

3) Trajectory Planning and Execution: RRT-Connect motion planner implementation is used for the Cartesian trajectory planning. It is one of the most efficient planners for the UR5 manipulator. *RRT-Connect* stands for Rapidly-exploring Random Trees (RRTs). The method works by incrementally building two Rapidly-exploring Random Trees (RRTs) rooted at the start and the goal configurations. The trees each explore space around them and also advance towards each other through the use of a simple greedy heuristic [25].

For the trajectory planner to work successfully, the obstacles are precisely modeled in the environment. Then, the free space is defined by calculating all the points, which can be successfully reached given that no part of the robot collides with the obstacles. Then, two the RRTs are initialised both, at the start and goal positions and grown in the free space. The exploration uses the randomly asssigned direction and magnitude of vectors, but they are biased towards the goal position as well as unexplored spaces. If the tree reaches the goal position, or meets the other tree grown from the opposite direction, the successful trajectory has been found. In our case, the exploration is continued until the planning time limit is reached, so more than one successful trajectory can be found. When one or more successful trajectories are found, they are smoothened to avoid choppy robot movements, and a total cost considering the the distance and danger zone crossings (using values from Eq. (4)) is calculated. The most efficient path is executed for the robot to successfully reach the goal position. A two dimensional example of *RRT-Connect* algorithm can be seen in Fig. 6.



(a) Exp. 1: End-effector position during robot movements between points A and B with reactive behaviour trajectory planning and a static obstacle. Static obstacle is indicated by the red zone.

(b) Exp. 2: End-effector position during robot movements between points A and B with a reactive behaviour trajectory planning and a dynamic obstacle. The obstacle was in the areas marked in red, yellow and green with high, medium and low frequency respectively.

(c) Exp. 3: End-effector position during robot movements between points A and B with our proposed method based on danger maps for trajectory planning and a dynamic obstacle. The obstacle was in the areas marked in red, yellow and green with high, medium and low frequency respectively.

Fig. 8. Comparison of our proposed method and reactive behaviour by tracking the end-effector trajectories under different conditions. The experiment was executed using the presented setup containing one UR5 robot and four 3D cameras. Resulting end-effector trajectories show that our method results in significantly shorter trajectories (compared to a reactive behaviour planning) by taking a calculated risk of crossing part of the danger zone when appropriate instead of taking a long traversal around the areas where a dynamic obstacle might be present. Actual trajectories were planned in 3D, however, for easier visualisation only a front view is shown here, where the difference between the presented methods is the most evident.

This method is suitable, because we can add our calculated cost function to the search space of *RRT-connect* to punish traversals passing through the risk areas calculated in Eq. (4).

Motion executions are performed using a new ROS implementation of a velocity based controller with rapid executions and smooth joint accelerations making the robot motions even more human-like. Robot control is done by calculating and directly sending speed commands to each of the robot joints, thus reducing the execution start time to 50-70 ms compared to around 170 ms using the traditional ROS UR5 drivers. The new controller is usable out-of-the-box and compatible with traditional *MoveIt!* trajectory execution [26].

IV. EXPERIMENTS AND RESULTS

To test our method, we split the experiments into two main parts. First, we evaluate the danger map construction by using a number of repetitive trajectories of an obstacle moving through the workspace. Secondly, the full system is evaluated by using both static and dynamic obstacles in the workspace. All the experiments were conducted using real hardware: the UR5 robot manipulator, two Kinect V2, one Kinect V1 and one Intel F200 3D cameras, as shown in Fig. 2. All the processing was done in real time two computers working in parallel connected with 1000 Mbit/s internal network.

A. Parameter Values

Algorithm parameter values were found by *trial-and-error* during the development of the presented method. They have proven to provide good accuracy, while still keeping the processing time low enough for fast and smooth execution of the robot movements. One set of parameter values was used in the current experiments and they are summarised in Table I.

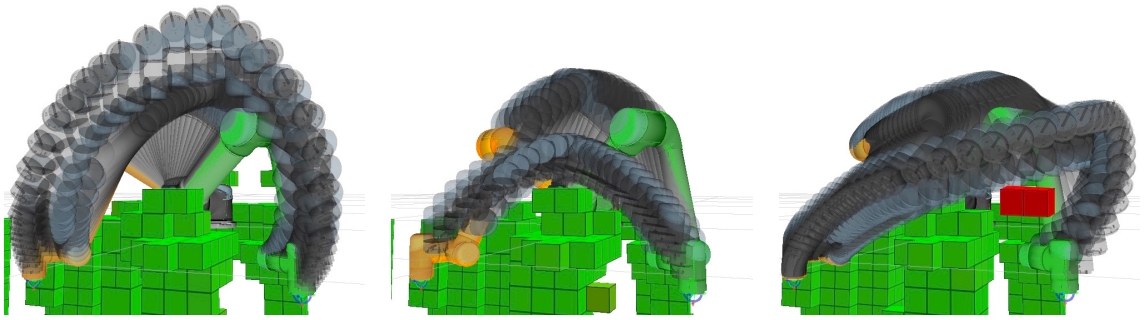
TABLE I. Parameter values used in our experiments.

Parameter	**Value**
Cost Function: α	0.3
Collision Risk Function: γ	1.5
Octomap Voxel Size	0.05 meters
Backtracking Move Magnitude	10% of the executed trajectory
Max Robot Joint Speed	50% of the maximum
Max Robot Joint Acceleration	60% of the maximum
Kinect V2 Refresh Rate	15 FPS
Kinect V1 Refresh Rate	15 FPS
Intel F200 Refresh Rate	30 FPS
RRT-Connect: Max Planning Time	1 sec
RRT-Connect: # Planning Attempts	3
RRT-Connect: Orientation Tolerance	0.1 rad
RRT-Connect: Position Tolerance	0.01 meter

B. Danger Map Construction

Danger map creation was tested separately by mounting a 10 cm by 20 cm object acting as an obstacle on the end effector of the robot and executing pre-defined trajectories in the workspace as shown in Fig. 7. The trajectories were close to a circular shape and executed in a continuous order one after another. The whole process was repeated 20 times. Trajectory 1 had a radius of $20cm$, trajectory 2 had a radius of $32cm$ and trajectory 3 had a radius of $46cm$. The execution time of each trajectory was 9.1 sec, 14.69 sec and 21.06 sec respectively. With the current cost function parameter α set to 0.3, the decay time of the voxel from occupied to free is $24.48sec$.

Throughout the process, the cost values C_{voxel} of all the voxels were tracked and averages calculated. This resulted in a complete danger zone, which could be sub-divided into three areas by the average costs representing the severity of possible collisions. As expected, the outside circle, had the lowest risk with values ranging between 0.0 and 0.2, the middle circle had



(a) Reactive behaviour: Planned trajectory when the obstacle is determined as static object blocking the direct path.

(b) Our method: Planned trajectory while avoiding the danger zone caused by obstacle occasionally entering the workspace.

(c) Our method: Planned trajectory while avoiding the danger zone and reflexive behaviour initialised by an obstacle (in red).

Fig. 9. Visualisation of the executed movements on the UR5 robot. Octomaps use the data from 3D cameras and different trajectory planning approaches were compared. Green robot shadow indicates the start position and yellow robot shadow indicates the goal position. Robot shadow trail indicates the trajectory.

a medium risk with values ranging between 0.2 and 0.5 and the inner circle had the highest average risk with values ranging between 0.5 and 1.0. The result was as expected, where the voxels falling in the inner area were occluded by robot's body when executing the outer trajectory 3. Results can be seen in Fig. 7 with all three trajectories marked and cube colors indicating the different average risk levels.

C. Operation of The Whole System

Operation of the whole system was tested by planning and executing the trajectory between the two pre-defined points A and B in the workspace. Our proposed method was compared against a simple reactive behaviour based on the same *RRT-Connect* trajectory planner which is used as a baseline. In the first experiment the reactive behaviour planner was used with a static obstacle blocking a direct path between points A and B. The second experiment the reactive behaviour planner was used with a dynamic obstacle randomly moving (moved manually by the operator) into the area blocking a direct path between points A and B. And in the third experiment, our proposed method, based on predictive and reflexive behaviour, was tested by using identical dynamic obstacle randomly moving into the area blocking a direct path between points A and B. In total, 15 executions of return A-B-A trajectories were executed in each of the experiments.

Because the *RRT-Connect* algorithm is based on random elements and provides different solution every time, the planned trajectories were different for every movement. In the experiment 1, the reactive behaviour planner successfully traversed around the static obstacle while keeping a safe margin between the robot and the obstacle, as seen in Fig. 8(a).

In the experiment 2, the obstacle randomly entered the indicated workspace with different frequency. Trajectories created by the reactive behaviour planner were significantly more random and spread all around the workspace. It was caused by some attempts of moving more or less directly between the two points and then reacting to a blocked path by the dynamic obstacle. In such cases, the robot stopped and quickly re-planned the trajectory. However, due to very limited planning time, the new trajectory was often not optimum and took a long and unnecessary detour with high safety margins. Longer allowed re-planning times would make new trajectories more optimal, however, the whole execution time would be likely to be even longer. Resulting trajectories of the experiment 2 can be seen in Fig. 8(b).

In the experiment 3, the dynamic obstacle was acting in an identical manner as in the experiment 2. Our proposed trajectory planner was constructing a danger map and using it to plan the trajectories between the points A and B, and the planned paths were significantly shorter and smoother compared to the results using the reactive behaviour. Low risk areas were often crossed with occasional crossings of the medium risk areas and a few crossings of the high risk areas. The reflexive behaviour was initialised only in two instances, both times the robot successfully moving away from the obstacle and avoiding the collision. On average, our proposed method provided the smoothest and shortest trajectories compared to the other two experiments. Resulting trajectories of the experiment 3 can be seen in Fig. 8(c).

Another evaluation criteria can be trajectory planning, optimisation and execution time. Our method demonstrated in the experiment 3 has over three times faster average trajectory execution time compared to the reactive behaviour (experiment 2) when the dynamic obstacle was present. The big difference mainly appears due to multiple stops by the reactive behaviour planner to replan the motion when the obstacle blocks the trajectory being executed. Normally it moves in the most direct free trajectory without considering the historical data, which in our method is considered by looking at the danger map. Furthermore, our proposed method is close to 20% quicker than the experiment 1, where the trajectory planning was performed to avoid the static obstacle. No significant differences were observed in planning and trajectory optimisation times between the three experiments, however our proposed method has the shortest time by a small margin. Timing results of all three experiments and a comparison against the obstacle-free *direct trajectory* can be found in Table II.

Some selected example trajectories of each method with the full UR5 body visualised are shown in Fig. 9 for easier

TABLE II. Trajectory planning and execution timing results containing averages and standard deviations of the path planning time, the path optimisation (smoothing) time and the path execution time. Results are compared against *direct trajectory*, which had no obstacles present.

	Exp. 1	**Exp. 2**	**Exp. 3**	**Direct**
Planning time (s)	0.0183 ±0.0099	0.0217 ±0.0095	0.0141 ±0.0072	0.0145 ±0.0057
Optim. time (s)	0.0178 ±0.0102	0.0130 ±0.0089	0.0129 ±0.0077	0.0006 ±0.0002
Execution time (s)	10.6085 ±4.0766	26.6663 ±14.7546	8.2952 ±4.3454	4.994 ±0.5421

visualisation of how the execution looks on the real system.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the predictive and reflexive trajectory planning method for a robot manipulator based on a danger map constructed using a multi 3D camera system. It is designed to function better in the workspaces where unknown dynamic obstacles are present. In the experiments, it has proven to be more effective than the traditional reactive trajectory planner, and still being able to avoid collisions with unexpected obstacles getting close to the robot body.

Our proposed system contains a combination of many methods working in parallel, each one of them having a set of tunable parameters, affecting the performance of the whole system. In our tests, a trial-and-error method was used to find a good combination of parameter values. However, in the future work, we plan to use AI systems, like evolutionary algorithms, to automatically compute the best parameter value set for case and make the system even more adaptive to changing conditions by learning over time.

Additionally, obstacle classification will allow the robot to react differently depending whether a person is approaching the robot, or some other object. Also, making a difference between bare hand or somebody holding a tool, especially a sharp one, a different size safety zone should be used and the robot should engage in different behaviour.

For collaborative tasks, the contact between the robot and human might be beneficial. With modeling and understanding the behaviour of a person sharing the workspace, joint tasks for object handover or robot working as a support as well as directing certain tools to a required area will become possible.

ACKNOWLEDGMENT

This work is partially supported by The Research Council of Norway as a part of the Engineering Predictability with Embodied Cognition (EPEC) project, under grant agreement 240862

REFERENCES

[1] I. Bonev, "Should We Fence the Arms of Universal Robots?" http://coro.etsmtl.ca/blog/?p=299, ETS, 2014, accessed August 10, 2016.

[2] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 338–345.

[3] P. Rakprayoon, M. Ruchanurucks, and A. Coundoul, "Kinect-based obstacle detection for manipulator," in *System Integration (SII), 2011 IEEE/SICE International Symposium on*. IEEE, 2011, pp. 68–73.

[4] C. Fitzgerald, "Developing baxter," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[5] J. Roach and M. Boaz, "Coordinating the motions of robot arms in a common workspace," *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 437–444, 1987.

[6] J. Leitner, S. Harding, M. Frank, A. Förster, and J. Schmidhuber, "Transferring spatial perception between robots operating in a shared workspace," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1507–1512.

[7] E. A. Merchán-Cruz and A. S. Morris, "Fuzzy-GA-based trajectory planner for robot manipulators sharing a common workspace," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 613–624, 2006.

[8] C. Breazeal, C. D. Kidd, A. L. Thomaz, G. Hoffman, and M. Berlin, "Effects of nonverbal communication on efficiency and robustness in human-robot teamwork," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 708–713.

[9] J. Y. Lew, Y.-T. Jou, and H. Pasic, "Interactive control of human/robot sharing same workspace," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2000, pp. 535–540.

[10] C. Lenz, M. Grimm, T. Röder, and A. Knoll, "Fusing multiple kinects to survey shared human-robot-workspaces," *Technische Universität München, Munich, Germany, Tech. Rep. TUM-I1214*, 2012.

[11] R. Hayne, R. Luo, and D. Berenson, "Considering avoidance and consistency in motion planning for human-robot manipulation in a shared workspace," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3948–3954.

[12] P. Fankhauser, M. Bloesch, D. Rodriguez, , R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *IEEE International Conference on Advanced Robotics (ICAR) (submitted)*, 2015.

[13] C. Amon, F. Fuhrmann, and F. Graf, "Evaluation of the spatial resolution accuracy of the face tracking system for kinect for windows v1 and v2," in *Proceedings of the 6th Congress of the Alps Adria Acoustics Association*, 2014.

[14] S. Foix, G. Alenya, and C. Torras, "Lock-in Time-of-Flight (ToF) Cameras: A Survey," *Sensors Journal, IEEE*, vol. 11, no. 9, pp. 1917–1926, 2011.

[15] Intel, "Specifications for the Intel RealSense Camera F200," https://communities.intel.com/docs/DOC-24012, Intel, 2015, accessed August 10, 2016.

[16] 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 *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[17] T. Wiedemeyer, "IAI Kinect2," https://github.com/code-iai/iai_kinect2, Institute for Artificial Intelligence, University Bremen, 2014 – 2015, accessed August 10, 2016.

[18] S. Ma and Z. Hu, "Hand-eye calibration," in *Computer Vision*. Springer, 2014, pp. 355–358.

[19] R. Horaud and F. Dornaika, "Hand-eye calibration," *The international journal of robotics research*, vol. 14, no. 3, pp. 195–210, 1995.

[20] J. Miseikis, K. Glette, O. J. Elle, and J. Tørresen, "Automatic calibration of a robot manipulator and multi 3d camera system," *CoRR*, vol. abs/1601.01566, 2016. [Online]. Available: http://arxiv.org/abs/1601.01566

[21] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Robotics-DL tentative*. International Society for Optics and Photonics, 1992, pp. 586–606.

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

[23] 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.

[24] Wikipedia, "Braking distance," https://en.wikipedia.org/wiki/Braking_distance, accessed August 10, 2016.

[25] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

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

III

PAPER III

3D Vision Guided Robotic Charging Station for Electric and Plug-in Hybrid Vehicles

J. Miseikis, M. Rüther, B. Walzel, M. Hirz and H. Brunner

OAGM/AAPR & ARW Joint Workshop 2017, Wien, Austria.
Nominated for the Best Student Paper Award

3D Vision Guided Robotic Charging Station for Electric and Plug-in Hybrid Vehicles

Justinas Mišeikis[1], Matthias Rüther[2], Bernhard Walzel[3], Mario Hirz[3] and Helmut Brunner[3]

***Abstract*— Electric vehicles (EVs) and plug-in hybrid vehicles (PHEVs) are rapidly gaining popularity on our roads. Besides a comparatively high purchasing price, the main two problems limiting their use are the short driving range and inconvenient charging process. In this paper we address the latter by presenting an automatic robot-based charging station with 3D vision guidance for plugging and unplugging the charger. First of all, the whole system concept consisting of a 3D vision system, an UR10 robot and a charging station is presented. Then we show the shape-based matching methods used to successfully identify and get the exact pose of the charging port. The same approach is used to calibrate the camera-robot system by using just known structure of the connector plug and no additional markers. Finally, a three-step robot motion planning procedure for plug-in is presented and functionality is demonstrated in a series of successful experiments.**

I. INTRODUCTION

Nowadays it is common to see electric vehicles and plug-in hybrids on our roads. Worldwide plug-in vehicle sales in 2016 were 773600 units, 42% higher compared to 2015 [1]. For example Norway plans to rule out sales of any combustion engine cars by 2025 [4]. However, a new problem being faced by EV and PHEV drivers is having an accessible, fast and convenient battery charging, especially when traveling longer distances. It is a common problem of fast chargers being idly occupied after the car is fully charged if the owner does not return to the vehicle. For example, Tesla has added an additional idle fee to discourage drivers leaving their cars at the chargers for longer than necessary [7]. A solution to avoid this problem and to enable a comfortable fast charging would be an automated robot-based charging system combined with automated car parking.

A. Charging Ports and Cables

Worldwide, there are many types of EV and PHEV charging ports, as well as different charging port placement locations on the vehicle. Each one of them has benefits and detriments, and car manufacturers have not decided on a common standard yet. This introduces an additional inconvenience of finding the correct type of charger, or having to carry a number of bulky adapters. As long as there is no standard, it would be more convenient to let the charging station detect the correct port type and adapt accordingly.

Another issue is the current weight and stiffness of a quick charging cable. For example, the weight of a CCS-Type 2 charging cable rated for the power up to 200 kW is 2.26 kg/m and outer diameter of 32 mm. With longer cable lengths, this becomes difficult for people to handle, but would not be an issue for a robot [6]. Cooled charging cables can help to solve this problem without increasing the cable diameter, but these are not yet standard [17].

B. Existing Automated EV Charging Methods

Automatic charging solutions have been researched both in academic and industrial environments. Volkswagen has presented an e-smartConnect system, where a Kuka LBR-iiwa robot automatically plugs in the vehicle after it autonomously parks in a specific target area (allowing for less than 20 cm by 20 cm error). It is also limited to one charging port type [8].

Tesla has demonstrated a concept of a snake-like robot automatically plugging in their EV, however, no technical details on the charging port localisation or robot operation were revealed [9].

The Dortmund Technical University has presented a prototype of the automatic charging system called ALanE. It is based on a robot arm capable of automatically plugging and unplugging a standard energy supply to an electric vehicle. The system is controlled via smartphone. However, full capabilities and flexibility of this concept system are not clear [3].

The NRG-X concept presents itself as a fully automatic charging solution. It can be adapted to any EV or PHEV and is capable of fast charging. Furthermore, it has a tolerance for inaccurate parking positions. The NRG-X system is based on combination of conductive and inductive charging on the under-body of the vehicle, thus an adapter for the vehicle is necessary. Furthermore, in the current concept configuration the charging power is limited to 22 kW [5], which results in over 7 times longer charging compared to 170 kW charging [22] and perspective 350kW [11].

[1]Justinas Mišeikis is with Department of Informatics, University of Oslo, Oslo, Norway justinm@ifi.uio.no

[2]Matthias Rüther is with Graz University of Technology, Institute for Computer Graphics and Vision, Graz, Austria ruether@icg.tugraz.at

[3]Bernhard Walzel, Mario Hirz and Helmut Brunner are with Graz University of Technology, Institute for Automotive Engineering, Graz, Austria {bernhard.walzel, mario.hirz, helmut.brunner}@tugraz.at

Comparisons of the time taken to charge a vehicle using different charging systems is shown in Figure 1.

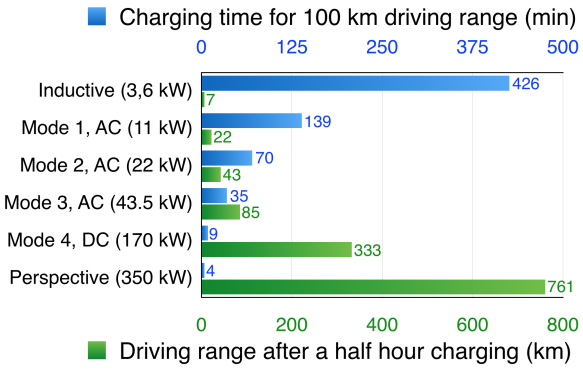


Fig. 1. Driving distance and charging time comparison of different charging systems [22].

C. Related Research

Automated charging has been well researched, especially for mobile robots. Typically, there is a custom made charging station, which is localized by the robot either using a direct communication or using computer vision based methods. These methods are normally based on having special markers on the charging station, which are localised in order for the robot to correctly align itself and approach the station. Removing markers would impede the operation [12] [19] [18] [14].

Another concept developed specifically for the detection of charging ports on EVs was based on adding an array of RFID tags on the car. Reading RFID signals allows to find the exact position and orientation of the charging port and plug it in automatically [16]. However, this still requires modification to the vehicle and would not support non-adapted cars.

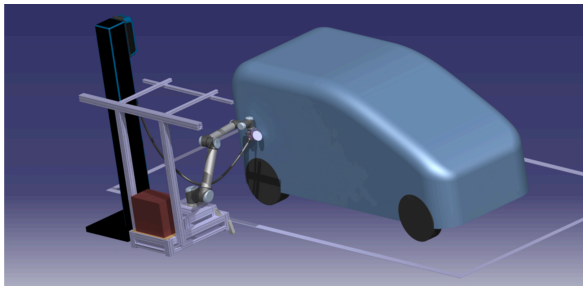


Fig. 2. CAD model of the robotic charging station concept.

D. Method Presented in This Work

We present a conductive robot-based automated charging method for EVs and PHEVs, which does not require any modifications to existing vehicles. First of all, we present a quick eye-to-hand calibration procedure to calibrate the vision sensor and the robot to work in the same coordinate system. It estimates both, the placement of the vision sensor in relation to the robot base as well as between the end-effector and the plug. Then we use shape-based matching and triangulation to locate and identify the charging port of the car and guide the robot, holding a charging cable, to precisely plug in the charger. Once the car is fully charged, the robot will automatically unplug from the vehicle, which will be ready to be driven away. The visualisation of the concept robotic charging station is shown in Figure 2.

This paper is organized as follows. We explain the proposed method in Section II. Then we provide our test setup, experiments and results in Section III, followed by conclusions and future work in Section IV.

II. METHOD

A. Detection of the Charging Port

A majority of the car charging ports are manufactured from texture-less black plastic material, making it difficult to obtain good features in the camera image. Similarly, the measurements made using time-of-flight cameras, which use the projection of infrared (IR) light, are noisy and inaccurate due to IR absorption by the material. As an alternative solution, a stereo-camera setup was used as the vision sensor.

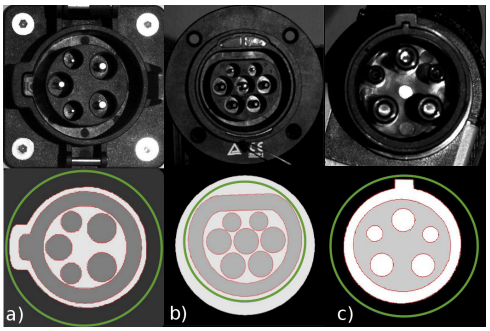


Fig. 3. Input images, simplified template models and automatically created shape-based templates for matching. Type 2 socket is shown in column a), type 1 socket in b) and type 2 connector plug is shown in c). Green circles define the area of interest for the model creation and the red outline line defines the created shape model.

The first step in the detection procedure is to find the location of the charging port in stereo images using shape-based template matching. Models were created for two types of the charging ports as well as the power plug connector, later to be used for eye-to-hand calibration. Figure 3 shows the camera images and simplified model images, which are used to automatically generate shape-based templates later to be used for matching. Template matching was performed using a *Halcon Machine Vision* software, which has proven to perform well in given conditions of low-contrast input images [2]. Matching results in a 2D Affine transformation matrix defining the template location in the image.

By taking x and y coordinates of the corresponding object points in images from each of the stereo

cameras, the depth information defined by z-axis can be calculated. The vision sensor in our setup has both stereo cameras fixed in relation to each other looking slightly inwards, with rotation around Y (vertical) axis. Solving Eq. 1 provides the real-world coordinates X, Y and Z of a point seen by the stereo cameras. Inputs (x_1, y_1) and (x_2, y_2) are the point coordinates in camera 1 and camera 2 respectively. Variable f is the focal length of the camera and b defines a baseline (distance) between the stereo cameras. Rotation between the cameras around Y-axis is defined by θ.

$$\begin{aligned} Z_0 &= \frac{b}{tan(\theta)} \\ Z &= \frac{b*f}{x_1 - x_2 + \frac{f*b}{Z_0}} \\ X &= \frac{x_1 * Z}{f} \\ Y &= \frac{y_1 * Z}{f} \end{aligned} \tag{1}$$

After the charging port is found in the input images, stereo triangulation is used to obtain 3D real-world coordinates of the port position, providing 5 to 7 reference points depending on the charging port type. Using the points, a perspective transformation is calculated using the least squares fit method to obtain the exact position and orientation of the charging port in relation to the vision sensor. Least squares fit for finding the orientation optimises for 3 unknowns (A, B and C), which later are mapped to roll, pitch and yaw angles. The least square error function is defined in Eq. 2, where x, y and z are coordinates of the reference points.

$$e(A,B,C) = \sum (Ax + By + C - z)^2 \tag{2}$$

Then, the error function is differentiated and set to zero, as shown in Eq. 3.

$$\begin{aligned} \frac{\partial e}{\partial A} &= \sum 2(Ax + By + C - z)x = 0 \\ \frac{\partial e}{\partial B} &= \sum 2(Ax + By + C - z)y = 0 \\ \frac{\partial e}{\partial C} &= \sum 2(Ax + By + C - z) = 0 \end{aligned} \tag{3}$$

The resulting linear equations with 3 unknowns are solved to get the orientation of the object. This can also be seen as 3D plane fitting to the given points.

B. Marker-less Eye-to-Hand Calibration

In order to operate the vision sensor and the robot in the same coordinate system, eye-to-hand calibration is necessary. The eye-to-hand calibration estimates the transformation between the vision sensor and the robot base. Using this transformation, the position of any object detected by the vision sensor can be recalculated into the coordinate system of the robot, allowing the robot to move to, or avoid that location.

Normally, a well structured object, like a checkerboard of known size and structure is used in the calibration process. However, it requires mounting it on the end-effector of the robot and can still result in additional offsets. We use the known structure of the connector plug and previously presented shape-based template matching with orientation estimation to obtain the precise pose. Eye-to-hand calibration is based on an automatic calibration procedure for 3D camera-robot systems, which uses the calibration method proposed by Tsai et al [15] [21].

The result of the eye-to-hand calibration are two transformation matrices. The first one defines the position of the vision sensor in relation to the robot base and the second one defines the position of the end point of the connector plug in relation to the end-effector of the robot.

The marker-less eye-to-hand calibration can be beneficial if the robot is placed on a moving platform, so the relative position between the vision sensor and the robot can change. Furthermore, it would benefit in cases when the robot has interchangeable end-effector attachments with different connector plugs. In both of these cases, recalibration procedure could be done automatically without any reconfiguration.

C. Robot Motion Planning

Given the limited workspace and all the movements being defined by camera measurements, robot control in Cartesian coordinates was used. The *MoveIt!* framework, containing multiple motion planning algorithms, was used for the initial testing [20]. The best performance in the defined case was demonstrated by the RRT-connect algorithm, which is based on the rapidly exploring random trees [13].

In order to get smoother motion execution and more human-like motions, a velocity based controller was used instead of the standard one provided in ROS. Better performance is achieved by calculating and directly sending speed commands to each of the robot joints, thus reducing the execution start time to $50-70$ ms compared to around 170 ms using the official ROS UR10 drivers [10].

D. Plugging-In Procedure

After the pose of the charging port is calculated, the coordinate system is assigned with the origin placed at the center of the plug and Z-axis looking outwards. Similarly, the coordinate system is assigned to the connector plug, which is held by the robot. The goal of the plug-in procedure is to perfectly align connector plug with the charging port, so the last movement is simply along one axis. In order to achieve that, a

three-step procedure was used, visualised in Figure 4. Firstly, the robot moves the plug at high velocity to the approach position, which is within a 0.1 meter radius from the charging port. The second step is to reduce the velocity to 10% of the maximum robot joint speed and move to the final alignment position. In this pose, the connector plug and the charging port are fully aligned by their Z-axis and just a few millimeters away from the contact point. The last step is to move at just 2% of the maximum speed along Z-axis and perform the plug-in motion. During this move, the forces and torques exerted on the end effector of the robot are monitored. In case the forces exceed a given threshold, the system is halted to prevent any damage.

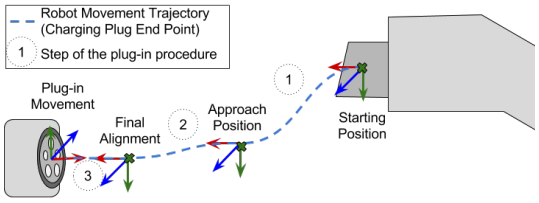


Fig. 4. Three step plug-in procedure plan. Firstly, the robot moves the connector plug to the *Approach Position*, which lies approximately 0.1 meter away from the charging port. The second move aligns the Z-axes of the charging port and the plug, and gets the plug just a few millimeters away from the port. The final plug-in movement performs the plugging in motion along Z-axis.

E. Unplugging

After the vehicle is charged fully or to the desired battery level, the robot has to disconnect the charger. Under the assumption that there were no position changes during the charging process, the unplugging procedure was simplified to follow the recorded way-points of the plug-in procedure in the inverse order. First, the robot gets back to the approach position and then returns to the stand-by position, where it is docked while waiting for the next task. The stand-by position ensures an unobstructed view of the parked vehicle for the vision sensor.

III. EXPERIMENTS AND RESULTS

A. Experiment Setup

At the current stage, the testing was limited to the lab environment. The experimental setup consists of an UR10 robot arm, a vision sensor containing stereo cameras and a charging port holder with inter-changeable charging ports. The charging port holder has variable height, position and angle to simulate various imperfect parking positions and differences in charging port locations on the vehicle. Two types of the charging ports, Type 1 and 2, have been used, as previously seen in Figure 3.

The connector plug is attached to the end-effector of the robot using a custom 3D printed attachment, shown in Figure 5. The charging cable is also attached

Fig. 5. Custom 3D printed connector plug holder attached to the end-effector of the UR10 robot.

to simulate realistic weight exerted on the robot during the operation. The whole experimental setup is shown in Figure 6.

The final goal was to locate the charging port using the vision sensor and estimate its pose. Then, the pose is transformed into the coordinate system of the robot and the end point of the connector plug is aligned and plugged in to the charging port. After a brief pause to simulate the charging process, the unplugging movement is performed and the robot moves back to the stand-by position.

Results of each part of the process are discussed separately and followed by the final evaluation of the whole system.

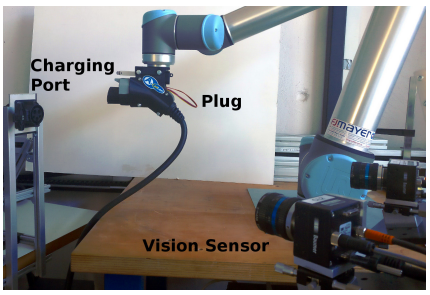


Fig. 6. The whole experiment setup. On the left the charging port holder can be seen. The robot is holding the connector plug, and the vision sensor made up of two stereo cameras is seen on the right hand side.

B. Template Matching

Template matching for Type 1 and Type 2 charging ports as well as the connector plug (Type 2) has worked well for various illumination and angles up to 45° relative to the viewing angle of the camera. The matching confidence score for good alignment was over 95%. The recognition speed on the full camera image was varying between 300*ms* and 800*ms*. By narrowing down the search area, for example by identifying the darker than average regions in the image, the recognition speed can be reduced to under 150*ms*. The results can be seen in Figure 7.

The limit for the successful recognition under low illumination or overexposure was when the edges of the socket or plug structure are still visible. The

connector plug was made out of more reflective plastic, resulting in a few cases when reflections caused the accuracy issues regarding the rotation. However, these issues were observed very rarely under specific viewing angles, and matching accuracy dropped below 90%, so these cases could be easily identified.

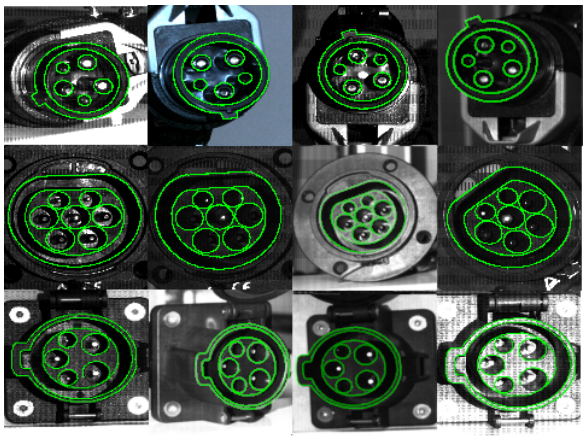


Fig. 7. Results of the template matching. A high variety of angles and lighting conditions were tested. Viewing angles up to 45° resulted in successful detection with accuracy dropping beyond that. Row 1: Type 2 connector plug. Row 2: Type 1 socket. Row 3: Type 2 socket.

C. Eye-to-Hand Calibration

In the given configuration, the structure of the connector plug was used as a marker for eye-to-hand calibration. During the calibration process it was turned to face the vision sensor, while during the normal operation it faces away from the camera. Furthermore, the outer ring of the plug is angled, so the pins of the plug had to be used as reference points to get the accurate calibration.

The end point of the connector plug was rotated around each of the axis as well as moved to different locations within the field-of-view of the vision sensor. In total, 26 poses were recorded and used until the calibration converged. Additionally, 3 instances were discarded because of the incorrect template matching result. The average translation error within the working space was reduced to 1.5*mm*, which was sufficient for our application at this stage. Possibly, having more poses would reduce the positional error even further. With the eye-to-hand calibration completed, coordinate frames for the camera position and the end point of the connector plug can be added to the model, as shown in Figure 8.

D. Finding Charging Port Pose and Robot Movements

As the final evaluation, we used the whole process pipeline and analysed whether the plug-in motion was successful or not.

There were 10 runs executed in total using Type 2 connectors. For the first 5 runs the charging port was

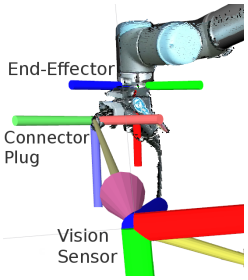


Fig. 8. Eye-to-hand calibration results. Visualisation of the assigned coordinate frames to the vision sensor, the end-effector of the robot and the end point of the connector plug. Resulting point cloud is overlayed onto the visualisation of the robot model.

angled at 10° in relation to the vision sensor, and for the remaining 5 runs, the angle was increased to 30°.

The robot successfully connected the plug 8 out of 10 times. Both failures occurred by missing the rotation of the plug, which were determined by the misalignment of the guidance slot on the charging port. However, the safety stop automatically initialised in both of the cases ensuring that the robot stopped before causing any damage.

TABLE I

SUMMARY OF THE PLUG-IN MOTION EXPERIMENTS WITH CHARGING PORT PLACED AT TWO DIFFERENT ANGLES

Exp	**Charging Port Angle** 10°	**Charging Port Angle** 30°
1	Success	Success: Misalignment
2	Success: Misalignment	*Failed: Missed rotation*
3	Success	Success
4	*Failed: Missed rotation*	Success: Misalignment
5	Success: Misalignment	Success: Misalignment

However, even when the plug was successfully inserted in the charging port, there were some alignment issues. In 5 out of 8 successful runs, the plug was not fully inserted into the charging port. It was caused by a small angular offset varying between 2° and 5°. The contact was still made, so the charging process would be successful, however, there was additional strain due to imperfect alignment. The misalignment occurred more frequently during the experiments, where the charging port was placed at 30° angle. The results are summarised in Table I.

As expected, the unplugging process was successful during all the runs. It simply follows already executed trajectory in the inverse order, meaning that as long as the position of the charging port did not change during the time it was plugged in, there should be no issues with the unplugging process.

IV. CONCLUSIONS AND FUTURE WORK

We have presented a vision-guided and robot-based automatic EV and PHEV charging station. The goal is to allow automated conductive fast charging of electric and hybrid vehicles and avoid the issue of a charged car taking up the space when it is not necessary.

The presented approach is a combination of multiple methods. First of all, the shape-based template matching is used to identify the charging port type and use the information from stereo cameras to precisely estimate its position and orientation. The same method is used in the marker-less eye-to-hand calibration, which results in the transformation matrices to be used to convert the position of the charging port from the coordinate system of the vision sensor to the robot. Then, the robot, holding a connector plug, is used to approach and finally plug in the charger cable into the EV or PHEV. Having a precisely estimated orientation is a big challenge and observation of the forces exerted on the end-effector of the robot are necessary to identify any possible misalignment, and stop or readjust if needed. Our approach has proven to work in the lab conditions under indoor illumination and using a custom made charging port holder.

Adding a force sensor to the robot would allow the robot to operate using the impedance controller based on force measurements and adjust it during the plug-in procedure according to the strains observed on the end effector. This would likely to be a solution for the observed cases with misalignment issues.

The project will be continued by improving the connector plug detection accuracy and automating the marker-less calibration procedure, where the robot would perform calibration movements automatically.

Furthermore, current tests were performed under the assumption that the charging port lid or cap was already opened. A linear actuator is already included in the setup, however, it was not used in current experiments. Future work includes finding the charger lid, identifying it's opening mechanism and using the robot to open and close it for the charging process. This would also require identification of the vehicle model to indicate the correct parking position and localise the approximate position of the charging port.

With the test electric vehicle to be delivered in the near future for testing purposes, the system will be evaluated on the real EV in the garage setup and outdoor tests. Communication between the vehicle and the charging station is also under development and this will enable the combination of the robot-based charging system with autonomous parking functions.

REFERENCES

[1] "EV-Volumes - The Electric Vehicle World Sales Database," http://www.ev-volumes.com/country/total-world-plug-in-vehicle-volumes/, (Accessed on 03/08/2017).

[2] "HALCON The power of machine vision - MVTec Software GmbH," http://www.mvtec.com/products/halcon/, (Accessed on 03/07/2017).

[3] "Ladesystem der TU Dortmund betankt Elektroautos automatisch - Fakultt fr Elektrotechnik und Informationstechnik - TU Dortmund," http://www.e-technik.tu-dortmund.de/cms1/de/Service_Termine/Weitere_Meldungen/Archiv/Ladesystem_Elektroautos/index.html, (Accessed on 03/03/2017).

[4] "Norway to completely ban petrol powered cars by 2025," http://www.independent.co.uk/environment/climate-change/norway-to-ban-the-sale-of-all-fossil-fuel-based-cars-by-2025-and-replace-with-electric-vehicles-a7065616.html, (Accessed on 03/08/2017).

[5] "NRG-X - Automatic Charging for E-Mobility," http://www.nrg-x.com/, (Accessed on 03/06/2017).

[6] "PHOENIX CONTACT — Homepage Corporate Website," https://www.phoenixcontact.com/, (Accessed on 03/07/2017).

[7] "Tesla owners who leave cars at Superchargers after charging will pay $0.40/minute," http://www.theverge.com/2016/12/16/13990854/tesla-supercharger-electric-fee-model-s-parking, (Accessed on 03/08/2017).

[8] "e-smartconnect: Volkswagen is conducting research on an automated quick-charging system for the next generation of electric vehicles," https://www.volkswagen-media-services.com/en/detailpage/-/detail/e-smartConnect-Volkswagen-is-conducting-research-on-an-automated-quick-charging-system-for-the-next-generation-of-electric-vehicles/view/2448500/7a5bbec13158edd433c6630f5ac445da, July 2015, (Accessed on 03/03/2017).

[9] "Tesla Unveils Snakelike Robot Charger for Electric Cars," http://www.livescience.com/51791-tesla-electric-car-robot-charger.html, 2015 August, (Accessed on 03/03/2017).

[10] T. T. Andersen, "Optimizing the Universal Robots ROS driver." Technical University of Denmark, Department of Electrical Engineering, Tech. Rep., 2015.

[11] C. Bracklo. (2016, Mar.) CharIN e.V.: The road to the success of a global charging standard - technology, standardization, organization. 2016. [Online]. Available: http://charinev.org/media/association-infos/

[12] U. Kartoun, H. Stern, Y. Edan, C. Feied, J. Handler, M. Smith, and M. Gillam, "Vision-based autonomous robot self-docking and recharging," in *Automation Congress, 2006. WAC'06. World*. IEEE, 2006, pp. 1–8.

[13] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[14] R. C. Luo, C. T. Liao, and K. C. Lin, "Vision-based docking for automatic security robot power recharging," in *Advanced Robotics and its Social Impacts, 2005. IEEE Workshop on*. IEEE, 2005, pp. 214–219.

[15] J. Miseikis, K. Glette, O. J. Elle, and J. Torresen, "Automatic calibration of a robot manipulator and multi 3D camera system," in *2016 IEEE/SICE International Symposium on System Integration (SII)*, Dec 2016, pp. 735–741.

[16] H. Oh, B. An, A. L. Smith, M. Raghavan, and F. C. Park, "An RFID localization algorithm for a plug-in electric vehicle recharging robot," in *Consumer Electronics (ICCE), 2015 IEEE International Conference on*. IEEE, 2015, pp. 176–177.

[17] PHOENIX CONTACT, "E-Mobility DC-Quickcharging with up to 350 A, online document," 2015. [Online]. Available: https://www.phoenixcontact.com/assets/downloads_ed/global/web_dwl_promotion/52007525_DE_INT_E-Mobility_LoRes.pdf

[18] M. Silverman, B. Jung, D. Nies, and G. Sukhatme, "Staying alive longer: Autonomous robot recharging put to the test," *Center for Robotics and Embedded Systems (CRES) Technical Report CRES*, vol. 3, p. 015, 2003.

[19] M. C. Silverman, D. Nies, B. Jung, and G. S. Sukhatme, "Staying alive: A docking station for autonomous robot recharging," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1. IEEE, 2002, pp. 1050–1055.

[20] I. A. Sucan and S. Chitta, "Moveit!" *Online at http://moveit.ros.org*, 2013.

[21] 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, 1989.

[22] B. Walzel, H. Brunner, and M. Hirz, "Requirements on Petrol Stations in Year 2025," in *14. Symposium Energyinnovation, Graz, Austria*, Feb 2016.

PAPER IV

IV

Robot Localisation and 3D Position Estimation Using a Free-Moving Camera and Cascaded Convolutional Neural Networks

J. Miseikis, P. Knobelreiter, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen

2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Auckland, New Zealand, 2018, pp. 181-187.
Finalist of the Best Student Paper Award

Robot Localisation and 3D Position Estimation Using a Free-Moving Camera and Cascaded Convolutional Neural Networks

Justinas Mišeikis[1], Patrick Knöbelreiter[2], Inka Brijacak[3], Saeed Yahyanejad[4], Kyrre Glette[5], Ole Jakob Elle[6], Jim Torresen[7]

***Abstract*— Many works in collaborative robotics and human-robot interaction focuses on identifying and predicting human behaviour while considering the information about the robot itself as given. This can be the case when sensors and the robot are calibrated in relation to each other and often the reconfiguration of the system is not possible, or extra manual work is required. We present a deep learning based approach to remove the constraint of having the need for the robot and the vision sensor to be fixed and calibrated in relation to each other. The system learns the visual cues of the robot body and is able to localise it, as well as estimate the position of robot joints in 3D space by just using a 2D color image. The method uses a cascaded convolutional neural network, and we present the structure of the network, describe our own collected dataset, explain the network training and achieved results. A fully trained system shows promising results in providing an accurate mask of where the robot is located and a good estimate of its joints positions in 3D. The accuracy is not good enough for visual servoing applications yet, however, it can be sufficient for general safety and some collaborative tasks not requiring very high precision. The main benefit of our method is the possibility of the vision sensor to move freely. This allows it to be mounted on moving objects, for example, a body of the person or a mobile robot working in the same environment as the robots are operating in.**

I. INTRODUCTION

Robotic manipulators are becoming cheaper resulting in new application fields outside the traditional industrial environment. It is more common to see robots in hospitals, warehouses and households. These environments result in robots having to share the workspace with people and even perform collaborative tasks. The concept of a shared workspace has been an active research area for many years, which is still highly relevant today [1] [2]. Furthermore, with the appearance of Industry 4.0, the need toward the environment and safety-aware robots is growing [3].

Collaborative robots, like Baxter and Sawyer, are advertised to be fully safe around people, however, it commonly means that they have sophisticated collision detection systems [4]. Ideally, collisions should be avoided at all, especially in sensitive environments like hospitals. Collision avoidance can be achieved by adding vision sensors.

Vision sensors observe the environment and indicate the areas which are unobstructed and safe to operate in, and are used to plan the robot movements accordingly. However, there are issues with this approach. Sensors have to be fixed on the robot itself or fixed in relation to the robot body. A precise Hand-Eye calibration is then performed to allow the sensors and the robot to operate in the same coordinate system. However, then the setup takes up more space and any unexpected disturbances or repositioning of the sensor can mean that the calibration has to be repeated. Despite automated calibration procedures, the process can still be cumbersome and time consuming [5]. Another option would be to fix the vision sensor on the robot body itself, commonly on the end-effector of the robot. This can be an effective method for collision avoidance for the end-effector of the robot, however, the field-of-view is normally limited and a full robot body collision check is rarely possible [6].

There has been a significant amount of work towards robot autonomy and self-localisation. However, robot self-awareness is normally limited to navigation, especially for mobile robots, or self-collision avoidance for robot arms or humanoid robots, where the robot model is known [7] [8] [9] [10].

Visual-based robot manipulator tracking has been extensively researched as well. End-effector being the main point of focus with the aim of conducting robot control based on visual servoing [11] [12]. Furthermore, it has proven to be an effective method for adaptive redundant robot control in Cartesian space [13]. Image-based tracking of 7-DoF robot arm showed promising results with dynamic parameter tuning as well [14]. Interesting work was presented, where authors are using particle swarm optimisation approach for fuzzy sliding mode control to track the end-effector of the robot manipulator [15]. Despite achieving good accuracy, most of these methods used prior knowledge or fixed setups for the particular use case. Changing the setup would result in the need to tune the algorithm for it to function accurately given the new conditions. Furthermore, commonly it was just the end-effector of the robot that was tracked instead of the whole robot body.

When looking at the field of human-robot interaction, a significant amount of work has been done on the design of the systems and workspaces allowing to monitor the human presence in close proximity to the robot and detect any irregularities [16] [17] [18]. Another work is focusing

[1 5 6 7]Justinas Mišeikis, Kyrre Glette, Ole Jakob Elle and Jim Torresen are with the Department of Informatics, University of Oslo, Oslo, Norway

[2] Patrick Knöbelreiter is with the Institute of Computer Graphics and Vision, Graz University of Technology, Graz, Austria

[3 4] Inka Brijacak and Saeed Yahyanejad are with the Joanneum Research - Robotics, Klagenfurt am Wörthersee, Austria

[6]Ole Jakob Elle has his main affiliation with The Intervention Centre, Oslo University Hospital, Oslo, Norway oelle@ous-hf.no

[1 5 7] {justinm,kyrrehg,jimtoer}@ifi.uio.no

[2] knoebelreiter@icg.tugraz.at

[3] Inka.Brijacak@joanneum.at

[4] Saeed.Yahyanejad@joanneum.at

Fig. 1. Samples from a collected robot dataset. Each row of images represents different robot type in the following order: UR3, UR5 and UR10. The dataset was created using a varying background to provide more robustness.

on the best approaches to safeguard the workspace of the robots [19].

When looking at the motion planning and behaviour prediction topics, most of the focus has been on modelling the human motions [20]. A heatmap of the workspace could be constructed to allow the robot to predict where dynamic obstacles are most likely to enter and have an additional reflexive behaviour override for unexpected cases [21].

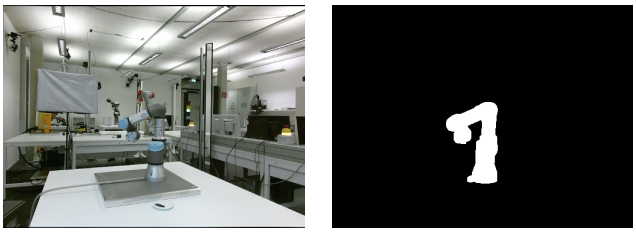
However, the majority of existing research has a robot model and control architectures well defined and fine-tuned. This means that the hardware setups are usually fixed and all the sensors have to be attached at the defined locations and calibrated specifically for the use case.

Our current research focuses on adding the flexibility on the robot identification side and allowing more unrestricted setups. For example, having a free-moving vision sensor as a part of the robotic system aimed at the robot safety or human-robot collaboration use case. We address this issue by removing the need for fixed setups. Instead of having a known transformation matrix between the coordinate frames of the sensor and the robot base, we teach the system to identify the robot body in a color image provided by the vision sensor. Our method uses convolutional neural networks (CNNs), which learn visual cues allowing it to understand the environment [22]. The system identifies the robot body in the color image of the vision sensor, and depth information normally provided by 3D cameras is not needed for the recognition task anymore. Furthermore, the system estimates the robot body configuration and 3D coordinates of each joint of the robot.

The vision sensor can be placed anywhere around the robot or even used as a wearable device by the robot operator. This approach can prove very useful in a cluttered environment where one or many robots are located, such as a factory floor or automated surgery room. Such environments can have a limited space for fixed camera setups or line-of-sight can be blocked by people or other machinery operating in close proximity. Having multiple free-moving cameras is one of the robust solutions ensuring the workspace is constantly observed. An operator or a visitor can have a wearable camera which observes the surroundings and indicates the positions of all the robots in the vicinity. A warning system or even an emergency stop option can be incorporated for the situations when the robot gets too close to the person to ensure a fully-safe operation.

Systems using our approach can also be useful in robot-robot interaction cases, where a mobile robot is operating in the same environment as robotic manipulators. It should avoid getting too close to other robots and avoid possible collisions. Even given a fully known environment, our system can prove useful if navigation or localisation algorithm fails to get an accurate position estimate, the vision sensor on the mobile robot can indicate positions where other robots are located. It can be useful for robot-to-robot interaction tasks. For example, if a mobile robot is bringing tools or objects that a robot arm needs to grasp, the mobile robot could localise itself in relation to the robot manipulator.

This paper is organized as follows. We present the system setup and dataset collection in Section II. Then, we explain the proposed method and CNN architecture in Section III. We provide experimental results in Section V, followed by relevant conclusions and future work in Section VI.



(a) Color Image (b) Ground truth model of the robot mask

Fig. 2. Example image of the dataset containing an UR3 robot.

II. SYSTEM SETUP AND DATASET COLLECTION

In our experiments, we use three types of robot arms produced by Universal Robots: UR3, UR5 and UR10. All three robots have a similar appearance, but different size and payload capabilities. A Kinect V2 camera was used

as a 3D vision sensor providing both color image and depth information, needed to create the dataset containing ground truth data [23]. The final, fully-trained system only needs the color image. The whole system was based on a combination of the Robot Operating System (ROS) and Theano framework [24].

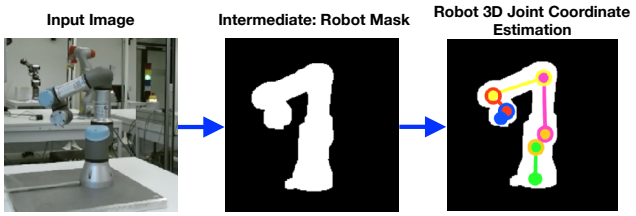


Fig. 3. Process described in regards to inputs and outputs of the system. A simple color image of the robot body is provided as an input to the system. The first CNN estimates the mask containing the robot body and this result is overlayed with the color image and used as an input to the second CNN. The second CNN provides an estimate of the joint coordinates of the robot in 3D. Each robot joint is visualised with a circle of a different color.

Deep learning requires a large amount of diverse training data to ensure efficient and robust learning. Given a limited availability of datasets for such applications, it was decided to create a dataset for this purpose. Access to the robots was obtained in three institutions: TU Graz, Joanneum Research and the University of Oslo.

In order to obtain a precise ground truth data, a Kinect V2 camera was placed at a number of positions overlooking the robot. At each position, a precise Hand-Eye calibration was performed by placing a marker on the end-effector of the robot and using both color and depth image for the calibration process [25]. Having a precise coordinate system transformation from the camera to the robot base allows us to know precisely where the robot is located in the camera image.

TABLE I. Dataset summary describing the number of samples collected for each type of the robot. In total 9 recordings were made, 3 for each type of the robot.

Recording	Robot Type	Number of Samples
Rec 1	UR3	211
Rec 2	UR3	252
Rec 3	UR3	463
Rec 4	UR5	252
Rec 5	UR5	756
Rec 6	UR5	1512
Rec 7	UR10	112
Rec 8	UR10	278
Rec 9	UR10	514

We used the MoveIt! package [26] to obtain ground truth data by using a robot self-filtering algorithm. At each time instance, the robot joints encoder information is combined with a simplified robot model, which is taken from the Unified Robot Description Format (URDF) file [27], to generate a precise estimation of the current robot pose in 3D space and a robot body mask as a 2D image. It can be used to find the robot in both, color and depth image.

Robot movements were pre-programmed in joints coordinate system to move in as many different configurations as possible without hitting an obstacle or self-collision occurring. Each of the robot joints is moved through the full range of motion in combination with other joints as well. The step size of the joint movements is varied between the datasets resulting in a different number of samples in each. This method ensured that the robot body will be observed from many angles by the vision sensor. After each movement was performed, a trigger signal was sent to record camera images, joint coordinates, Cartesian coordinates of each joint and ground-truth model of the robot position. The number of samples per dataset varied from 112 to 1512. In total nine datasets were collected, three for each type of the robot, summarized in Table I. Example images from the collected dataset can be seen in Figure 1. Datasets with the UR5 robot were the most extensive given the access to the robot at the lab of the main author. An example of color and ground truth images can be seen in Figure 2.

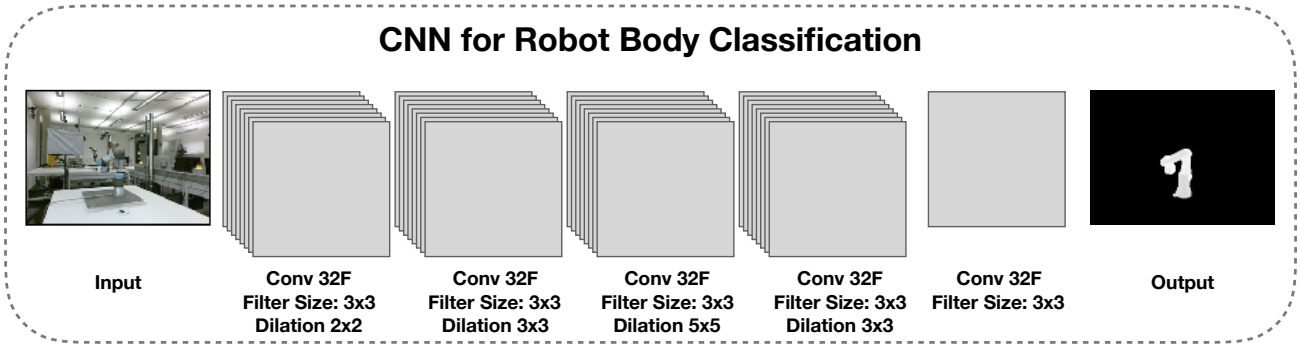
All the collected datasets were used for the training process, resulting in 926 samples for UR3, 2520 samples for UR5 and 904 samples for UR10. The datasets were split into training and test set by randomly assigning 80% and 20% of the images accordingly. All of the images have the resolution of 512×424 pixels and are rectified using an internal camera calibration to remove the lens distortion. Higher resolution, 960×540 pixels color images were recorded as well, however, in practice, we scaled and cropped images to have the same resolution for all the types: color, depth and ground truth mask to avoid any scaling issues.

III. METHOD

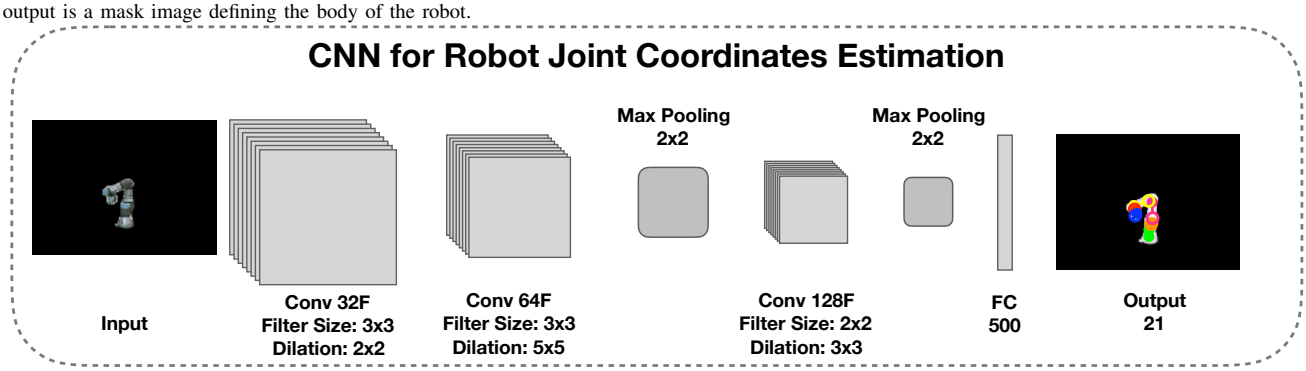
Our method is based on a two-level cascaded CNN, where one CNN is used for the classification task in foreground/background detection of the robot body in the image, and the second CNN is used for landmark detection of the robot joint positions in 3D coordinates. The process in regards to the input and output images is shown in Figure 3.

The principle of CNN is to have an image as an input, which is passed to the network. Normally, CNN contains a number of hidden layers, which lead to the output, which is also given during the training process, defined as ground truth. In the hidden layers, the network is capable of learning a number of filters, which help to achieve the desired result, thus minimising the error between the output of the network and provided ground truth result. The learning process is done by initialising random weights, getting the output, comparing it to the desired result and then adjusting the weights in the hidden layers during the back-propagation process in order to achieve better accuracy.

For the robot body classification, our CNN consists of four convolutional layers with 32 filters each and varying dilation was used as well as the last convolutional layer containing just one filter. The details about the architecture are illustrated in Figure 4(a). The loss function was specifically designed to take into consideration the rather small area of the foreground object in the input image. In most of the cases, the area of the robot body in the input image was $6 - 17\%$ of the whole image. Without this adjustment, in some cases,



(a) CNN architecture for the robot mask classification. The network consists of 5 convolutional layers with varying dilation. Input is a color image and output is a mask image defining the body of the robot.



(b) CNN architecture for the robot joint coordinate estimation. The network consists of 3 convolutional layers, 2 pooling layers and a fully connected layer in the end. Input is an overlayed color image with a robot foreground mask and output is 3D coordinates of the robot joints in the coordinate system of the vision sensor.

Fig. 4. Cascaded CNN architecture for the robot position estimation using non-fixed camera.

the network classifying all the pixels as background could still achieve the accuracy of over 90%, which is conceptually wrong. Calculation of the foreground weight w_{fg} is shown in Equation 1. It is based on the inverse probability of the foreground and background classes, where $Y \in \{fg, bg\}$.

$$w_{fg} = \frac{1}{\mathsf{P}(Y = fg)} \tag{1}$$

Similarly, the background weight w_{bg} is calculated using Equation 2.

$$w_{bg} = \frac{1}{\mathsf{P}(Y = bg)} \tag{2}$$

The loss function is calculated by first getting loss per pixel and then using it to calculate loss of the whole image. A normalisation factor $\mathcal{N}$, which is a number of pixels in the image, allows us to keep the learning rate fixed, independent of the image size.

Loss function for one pixel l^n is defined in Equation 3, where i_{est} is $\mathsf{P}(Y = fg)$, $(1 - i_{est})$ is $\mathsf{P}(Y = bg)$ and i_{gt} is the ground truth value from the mask image.

$$\begin{aligned} l^n(I^n_{est}, I^n_{gt}) = &- w_{fg} i_{est} \log(i_{gt}) \\ &- w_{bg}(1 - i_{est}) \log(1 - i_{gt}) \end{aligned} \tag{3}$$

The result is then used to calculate normalised loss for the whole image $\mathcal{L}_{mask}$ using Equation 4.

$$\mathcal{L}_{mask}(I_{est}, I_{gt}) = \frac{1}{\mathcal{N}} \sum_n l^n(i_{est}, i_{gt}) \tag{4}$$

We formulate the joint coordinate estimation as a regression task using the second CNN. The network consists of three dilated convolutional layers with 32, 64 and 128 filters respectively, two max-pooling layers in between and a fully connected layer in the end. Details of the network architecture can be seen in Figure 4(b).

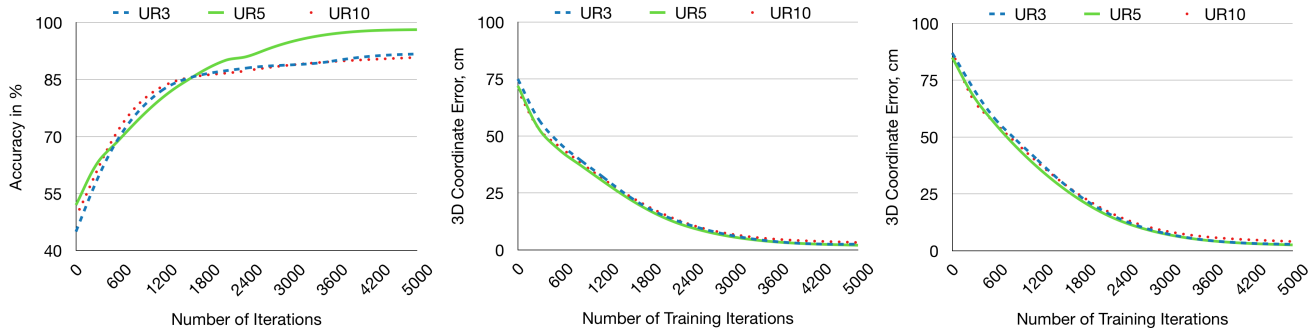
Loss function $\mathcal{L}_{coords}$ is based on Euclidean distance calculations between estimated and ground truth values as defined in Equation 5, where N_j is the number of joints, J_i defines ground truth position of each joint and E_i is the estimated position of each joint by CNN.

$$\mathcal{L}_{coords} = \frac{1}{N_j} \sum_{i=1}^{N_j} \|J_i - E_i\|_2 \tag{5}$$

In this work we decided not to use any prior robot model information to keep the system more adaptable to other robot models in the future, meaning that a raw CNN output is used to evaluate the accuracy of the results without any additional post-processing.

IV. CNN TRAINING

There are two main possibilities on how to train the cascaded CNN. The first option is to train the whole network



(a) Results: evaluation of the CNN trained for the robot body classification, mask accuracy over a number of iterations for all three robots using validation sets. It can be seen that UR5 outperformed UR3 and UR10.

(b) Results: evaluation of the CNN trained for 3D coordinate estimation of the robot joint positions using input based on ground truth mask data.

(c) Results: evaluation of the CNN trained for 3D coordinate estimation of the robot joint positions using the full system.

Fig. 5. Evaluation of our method based on accuracy over a number of training iterations.

end-to-end and observe the middle layer of the mask. However, this might not result in the output that is expected and is unlikely to reach the desired mask accuracy. In this work, we train each of the CNNs separately optimising for the best result at each stage. This approach provides the flexibility of using just a part of the system, for example, if only a mask for the robot body is needed. When running the full cascade, the output of the first CNN is used to mask a color input image and use it as an input for the second CNN. The training has been done on each type of the robot separately, however, by observing intermediate-level feature maps, we have noticed very similar features for all of the robot models.

The training of the classification CNN took slightly more than 2 days on a regular nVidia GeForce 1080 GTX graphics card for all the datasets. The data was selected in a random order to avoid any biases and split in mini-batches of 128 images each for input to avoid overloading memory of the GPU. The learning rate was gradually decreasing, starting at 0.001 and reducing to 0.000001 throughout the learning process. It took 5000 epochs for the network to converge. The input size was half of the original image size: 256×212 pixels. The pixel intensity values were converted to float and normalised to lay between 0 and 1. Additionally, pixel values of the ground truth image are clipped to avoid division by zero in cases when the estimated mask fits the ground truth perfectly.

Training of the coordinate estimation CNN was significantly faster, taking under 7 hours, also converging after 5000 epochs. The learning rate was adjusted during the training, starting at 0.03 down to 0.0001, and momentum was increased over epochs from 0.9 to 0.999.

V. RESULTS

For evaluation, we use test sets and analyze outputs of the trained systems against the ground truth data and calculate the accuracy of the system. For the robot body classification, the accuracy is defined by comparing how many pixels in the CNN output mask image match the ground truth mask. For the robot joint coordinates estimation, the error is defined by the Euclidean distance between the estimated coordinates and ground truth in all three dimensions, averaged over all joints of the robot.

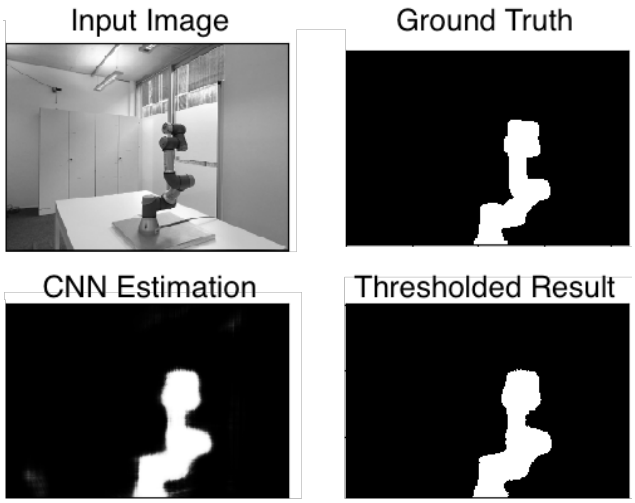


Fig. 6. An example result of the UR3 robot body mask classification including input, ground truth, raw and thresholded CNN output images. It can be seen that the mask fit corresponds well with the ground truth. The only drawback is that the fit is not as sharp as the ground truth image. However, no unwanted artefacts or false positives are present.

First, the results are presented for each of the CNNs separately and then of the whole system altogether. Results are analysed separately for the three types of robots.

A. Evaluation of the Robot Classification Task

First, we present the results of the robot classification task for each type of the robot. Accuracy is defined by the number of correctly classified pixels in the mask image. Classification of UR5 reached the accuracy of $98,1\%$ and outperformed UR3 and UR10 with $93,1\%$ and $92,8\%$ respectively. The accuracy results over the training iterations can be seen in Figure 5(a). An example mask estimation is shown in Figure 6.

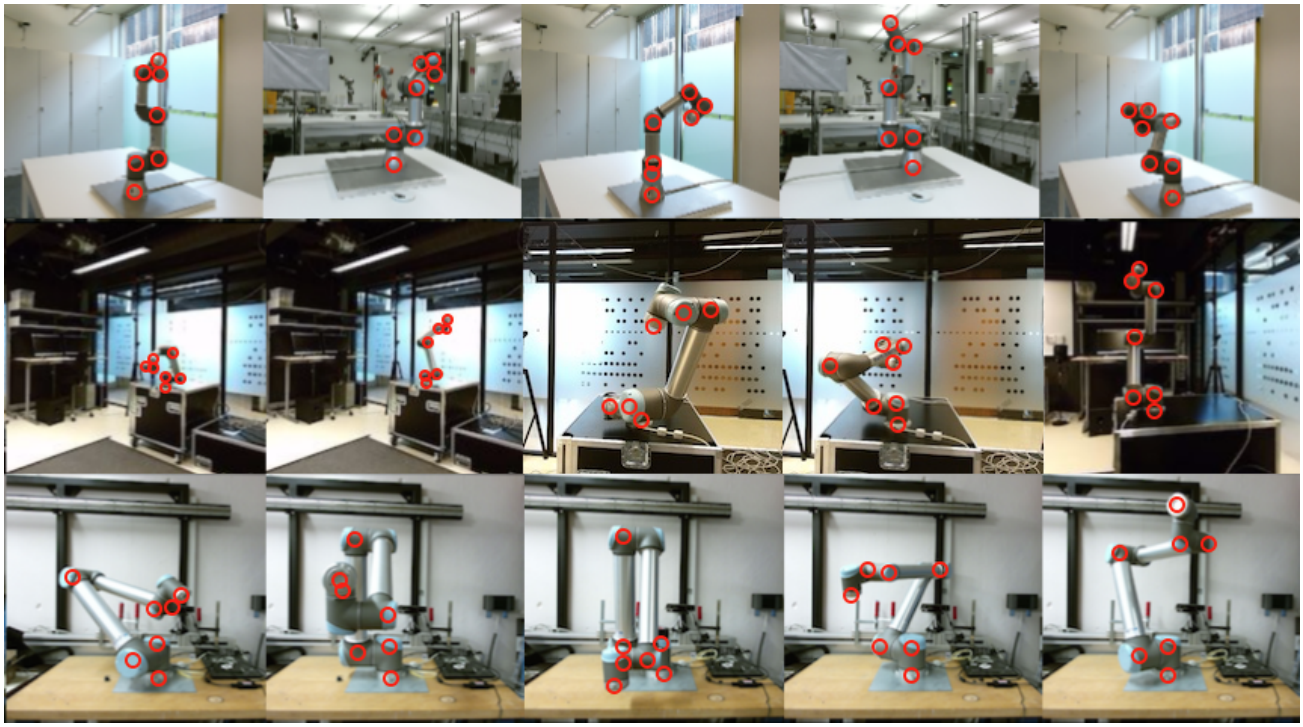


Fig. 7. Estimated robot joint position coordinates marked on the images taken from the dataset. Due to difficulty in visualising 3D coordinates on printed figures, the estimated joint coordinates were mapped back into 2D images. Each row represents UR3, UR5 and UR10 robots respectively.

B. Evaluation of the Robot Joint Coordinate Estimation

In order to analyse the coordinate estimation, first, we use the overlay images based ground truth mask data for the input. As expected, CNN trained on UR5 data provided the most accurate estimation with the average position error of $2,0cm$, while UR3 had the error of $2,5cm$ and UR10 - $3,2cm$. The coordinate estimation results over the training iterations can be seen in Figure 5(b).

C. Evaluation of the Full System

For the full system evaluation, the process is the combination of the previous two methods joined together: the resulting output image of the robot mask classification is used to overlay the color image and passed as an input for the robot joint coordinate estimation. It is imperfect compared to the ground truth data, so worse results were expected compared to the previous test. For the full system, the final coordinate estimation error increased to $2,4cm$ for UR5, $2,6cm$ for UR3 and $3,9cm$ for UR10. Results can be directly compared with the previous Section V-B.

The final results are summarised in Table II and the estimated coordinates by the full system marked over the dataset images can be seen in Figure 7. Because it is difficult to show 3D estimations on 2D figures, the visualisation of estimation is done by mapping the estimated 3D coordinates back onto input images.

TABLE II. Experiment Results Summary

	UR3	UR5	UR10
Mask Accuracy, %	$93,1\%$	$98,1\%$	$92,8\%$
Coordinates Error (separate)	$2,5cm$	$2,02cm$	$3,21cm$
Coordinates Error (full system)	$2,57cm$	$2,42cm$	$3,89cm$

VI. CONCLUSIONS AND FUTURE WORK

In this work, we have addressed robots for collaboration and human-robot interaction tasks. We have found an alternative solution for Hand-Eye calibration and added the flexibility of placing the camera at arbitrary position observing the robot workspace, while still being able to identify the robot in the image and estimate its position.

Our system uses a cascaded convolutional neural network to achieve the goal. For training and testing purposes, we have collected a number of datasets using a line of robots produced by Universal Robots: UR3, UR5 and UR10. This allowed us to precisely train the CNN and achieve the accuracy in robot body classification of up to 98% on the test set and 3D joint coordinate estimation with an error of less than $3cm$. Furthermore, we have shown that the accuracy directly correlates with the training duration and a number of collected samples. This result is still not accurate enough for applications like visual servoing, but it can be good enough for some collaboration tasks as well as safety alerts in cases where a person does not have to work in a very close proximity to the robot.

Some example applications would be a small body-mounted camera for doctors working in robotised operating rooms or visitors on the factory floor. It would also be useful in robot-robot interaction cases, when a mobile robot is operating in the same areas as the robot arms, either in order

to avoid each other, or support the operations by bringing objects, which would be handled by robot manipulators. The system would be trained to identify all the robots existing in the specific environment, and the person would be warned by a visual or audible alert in cases where he gets within the reachable distance of the robot. Furthermore, if the robot gets too close to the person, an emergency stop could be initiated.

For the human-robot collaboration tasks, hand tracking of a person can be achieved using devices like Leap Motion or skeleton tracking to get an estimate of the relative hand positions to the robot. This makes it possible to achieve the tasks like tool handover between the person and the robot, completing joint tasks or even hand-gesture control, while avoiding any unwanted physical contact between the two.

Further work includes expanding our method to new types of robots by using transfer learning from pre-trained CNN. This could allow achieving good accuracy with a limited number of training samples. Furthermore, we will add state-of-the-art skeleton tracking and human motion prediction to perform collaborative human-robot tasks and evaluate the performance compared to the cases of having fixed camera-robot setups.

ACKNOWLEDGMENT

This work is partially supported by The Research Council of Norway as a part of the Engineering Predictability with Embodied Cognition (EPEC) project, under grant agreement 240862, and by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) within the project framework CollRob (Collaborative Robotics).

REFERENCES

[1] J. Roach and M. Boaz, "Coordinating the motions of robot arms in a common workspace," *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 437–444, 1987.

[2] J. Leitner, S. Harding, M. Frank, A. Förster, and J. Schmidhuber, "Transferring spatial perception between robots operating in a shared workspace," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1507–1512.

[3] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[4] C. Fitzgerald, "Developing Baxter," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[5] J. Miseikis, K. Glette, O. J. Elle, and J. Torresen, "Automatic calibration of a robot manipulator and multi 3d camera system," in *System Integration (SII), 2016 IEEE/SICE International Symposium on*. IEEE, 2016, pp. 735–741.

[6] G. Flandin, F. Chaumette, and E. Marchand, "Eye-in-hand/eye-to-hand cooperation for visual servoing," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 3. IEEE, 2000, pp. 2741–2746.

[7] S. Schneegans, P. Vorst, and A. Zell, "Using RFID Snapshots for Mobile Robot Self-Localization." in *EMCR*, 2007.

[8] J.-S. Gutmann and C. Schlegel, "Amos: Comparison of scan matching approaches for self-localization in indoor environments," in *Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on*. IEEE, 1996, pp. 61–67.

[9] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar, "Real-time (self)-collision avoidance task on a hrp-2 humanoid robot," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3200–3205.

[10] A. De Santis, A. Albu-Schaffer, C. Ott, B. Siciliano, and G. Hirzinger, "The skeleton algorithm for self-collision avoidance of a humanoid manipulator," in *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*. IEEE, 2007, pp. 1–6.

[11] W. J. Wilson, C. W. Hulls, and G. S. Bell, "Relative end-effector control using cartesian position based visual servoing," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 684–696, 1996.

[12] A. Ruf, M. Tonko, R. Horaud, and H.-H. Nagel, "Visual tracking of an end-effector by adaptive kinematic prediction," in *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, vol. 2. IEEE, 1997, pp. 893–899.

[13] B. Daachi and A. Benallegue, "A neural network adaptive controller for end-effector tracking of redundant robot manipulators," *Journal of Intelligent & Robotic Systems*, vol. 46, no. 3, pp. 245–262, 2006.

[14] I. Siradjuddin, L. Behera, T. M. McGinnity, and S. Coleman, "Image-based visual servoing of a 7-DOF robot manipulator using an adaptive distributed fuzzy PD controller," *IEEE/ASME Transactions on Mechatronics*, vol. 19, no. 2, pp. 512–523, 2014.

[15] M. R. Soltanpour and M. H. Khooban, "A particle swarm optimization approach for fuzzy sliding mode control for tracking the robot manipulator," *Nonlinear Dynamics*, vol. 74, no. 1-2, pp. 467–478, 2013.

[16] G. Michalos, S. Makris, P. Tsarouchi, T. Guasch, D. Kontovrakis, and G. Chryssolouris, "Design considerations for safe human-robot collaborative workplaces," *Procedia CIrP*, vol. 37, pp. 248–253, 2015.

[17] T. B. Sheridan, "Human–robot interaction: status and challenges," *Human factors*, vol. 58, no. 4, pp. 525–532, 2016.

[18] I. Brijacak, S. Yahyanejad, B. Reiterer, and M. Hofbaur, "Toward safe perception in human- robot interaction," in *OAGM and ARW Joint Workshop, Vienna*. IEEE, May 2017, pp. 80–85.

[19] Y. Yamada, Y. Hirasawa, S. Huang, Y. Umetani, and K. Suita, "Human-robot contact in the safeguarding space," *IEEE/ASME transactions on mechatronics*, vol. 2, no. 4, pp. 230–236, 1997.

[20] J. Mainprice and D. Berenson, "Human-robot collaborative manipulation planning using early prediction of human motion," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 299–306.

[21] J. Mišeikis, K. Glette, O. J. Elle, and J. Torresen, "Multi 3D camera mapping for predictive and reflexive robot manipulator trajectory estimation," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–8.

[22] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[23] P. Fankhauser, M. Bloesch, D. Rodriguez, , R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *IEEE International Conference on Advanced Robotics (ICAR) (submitted)*, 2015.

[24] 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 *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[25] T. Heikkilä, M. Sallinen, T. Matsushita, and F. Tomita, "Flexible hand-eye calibration for multi-camera systems," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2000, pp. 2292–2297.

[26] I. A. Sucan and S. Chitta, "MoveIt!" *Online Available: http://moveit.ros.org*, 2013.

[27] W. Meeussen, J. Hsu, and R. Diankov, "URDF-Unified Robot Description Format," 2012.

PAPER V

Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images

J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle and J. Torresen.

2018 15th International Conference on Ubiquitous Robots (UR), Honolulu, HI, USA, 2018, pp. 597-603.

V

Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images

Justinas Mišeikis[1], Inka Brijacak[2], Saeed Yahyanejad[3], Kyrre Glette[4], Ole Jakob Elle[5], Jim Torresen[6]

***Abstract*— The field of collaborative robotics and human-robot interaction often focuses on the prediction of human behaviour, while assuming the information about the robot setup and configuration being known. This is often the case with fixed setups, which have all the sensors fixed and calibrated in relation to the rest of the system. However, it becomes a limiting factor when the system needs to be reconfigured or moved. We present a deep learning approach, which aims to solve this issue. Our method learns to identify and precisely localise the robot in 2D camera images, so having a fixed setup is no longer a requirement and a camera can be moved. In addition, our approach identifies the robot type and estimates the 3D position of the robot base in the camera image as well as 3D positions of each of the robot joints. Learning is done by using a multi-objective convolutional neural network with four previously mentioned objectives simultaneously using a combined loss function. The multi-objective approach makes the system more flexible and efficient by reusing some of the same features and diversifying for each objective in lower layers. A fully trained system shows promising results in providing an accurate mask of where the robot is located and an estimate of its base and joint positions in 3D. We compare the results to our previous approach of using cascaded convolutional neural networks.**

I. INTRODUCTION

With the tendency of robotic hardware becoming cheaper and more powerful, robots are entering our everyday environments. Household robots like vacuum cleaners do not surprise people anymore. Even faster robot adoption happens in hospitals, warehouses and factories. An important reason for this is advancements in environment perception capabilities. Instead of fencing off the robots, the concept of Industry 4.0 is aimed at having a new era of collaborative robots, which are safe to operate in shared workspaces with humans [1]. The concept of a shared workspace has been an active research area for many years, which is still highly relevant today [2] [3]. The industry is catching up to research with robotic platforms like Baxter and Sawyer, which are known to be fully safe to operate around humans. However, they are still at a stage, where collision detection is the main safety system [4]. But we are looking at more sensitive environments, for example, hospitals, where collision detection is not good enough and full avoidance is needed.

[1 4 5 6]Justinas Mišeikis, Kyrre Glette, Ole Jakob Elle and Jim Torresen are with the Department of Informatics, University of Oslo, Oslo, Norway

[2 3] Inka Brijacak and Saeed Yahyanejad are with the Joanneum Research - Robotics, Klagenfurt am Wörthersee, Austria

[5]Ole Jakob Elle has his main affiliation with The Intervention Centre, Oslo University Hospital, Oslo, Norway oelle@ous-hf.no

[1 4 6] {justinm,kyrrehg,jimtoer}@ifi.uio.no

[2] Inka.Brijacak@joanneum.at

[3] Saeed.Yahyanejad@joanneum.at

One of the most common methods to observe the environment is by using vision sensors. In this application, 3D cameras observe the workspace and indicate the areas, which are free of obstructions and are safe to operate in as well as obstacles, which should be avoided. Given this information, a robot can find the safest path to reach its goal. However, normally these sensors are fixed either in relation to the robot or in the environment. In order to function in the same coordinate frame and provide accurate information to the robotic system, Hand-Eye calibration is performed. It works well as long as the setup of the sensors and the robot base stays static. If any of them are moved, intentionally or accidentally, the calibration has to be repeated in order for the sensors to work with the necessary precision. Despite some automatic calibration procedures, the process can still be time-consuming, and the system has to be halted until this issue is resolved [5].

One way to make the environment aware robots is to use long-term environment observation. Such approaches have been used in the development of robot autonomy and self-localisation tasks. This is commonly developed as navigation algorithms for mobile robot platforms to find their way around in the environment and avoid any static or dynamic obstacles on the way. Typically, robot model and dynamics are typically known [6] [7] [8] [9].

Visual-based robot manipulator tracking has been extensively researched as well. End-effector being the main point of focus with the aim of conducting robot control based on visual servoing [10] [11]. Furthermore, it has proven to be an effective method for adaptive redundant robot control in Cartesian space [12]. Image-based tracking of 7-DoF robot arm showed promising results with dynamic parameter tuning as well [13]. In another project, authors use particle swarm optimisation method for fuzzy sliding mode control to track the end-effector of the robot manipulator [14]. Furthermore, robotic arms were combined with deep learning approaches to learn direct motor commands by using visual inputs. They were based on reinforcement learning and by trying thousands of grasps reaching impressive results of adaptive grasping approaches. However, that required many hours of training while using real hardware [15] [16] [17].

One thing that majority of discussed systems have in common is that prior knowledge of the robotic platforms is given or the setups in regards to hardware are fixed. Any changes to the setup would require re-calibration or at least fine-tuning the algorithms to achieve the same level of performance. Furthermore, common obstacle avoidance algorithms for robotic arms are focusing on the end-effector

Fig. 1. Samples from a collected robot dataset. Each row of images represents different robot type in the following order: UR3, UR5 and UR10. The dataset was created using a varying background to provide more robustness.

instead of the whole robot body.

Having non-fixed setup allows easier camera placement in cluttered environments with multiple robots, like a factory floor or automated surgery room. Normally, there is limited space and equipment might have to be shifted around quite frequently. This results in limited line-of-sight or people standing in front of the sensor. Having a multi-camera setup can add the needed redundancy, or using a wearable camera would provide a viewpoint of the operator. On a factory floor, such a camera-based system can give an indication of all the robots located around the person wearing it. A warning or even an emergency stop option can be incorporated into the system for the situations when the robot gets too close to the person within its field of view to ensure a safe operation.

A similar approach could be also used in robot-robot interaction cases, where similar or heterogeneous robots are working in the same environment. Even without having direct communication channels, robots can avoid collisions with each other. On the other hand, this can be used as a redundant navigation system, given the map of the main robots is known, the mobile platform can re-localise itself according to their detected positions. Collaborative tasks would be targeted also, where robots have to hand over tools or work together. Having an active communication channel is not always reliable, so being able to identify robot arms in the environment and their configuration using on-board camera can allow to solve these problems. Provided high enough processing power, swarm robotics could benefit from such systems, where each individual is making independent decisions without any centralised system.

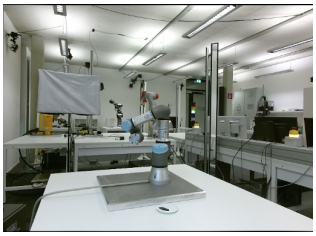
Our current research targets this problem by trying to add flexibility to the robot identification and having easily adjustable setups. One goal is to have a free moving camera and remove the need for Hand-Eye calibration. Instead of having a known transformation matrix between the coordinate frames of the sensor and the robot base, we teach the system to identify the robot body in a 2D color image provided by the vision sensor. This would allow having cameras placed on moving objects, for example, wearable ones or placed on other robots moving in the environment. Our method uses convolutional neural networks (CNNs), which learn visual cues allowing it to understand the environment [18]. The system identifies the robot body in the color image, and depth information normally provided by 3D cameras is not needed for the recognition task anymore. Furthermore, the system estimates the robot body configuration and 3D coordinates of each joint of the robot.

Current work is an extension and improvement of our previously proposed method to use cascaded CNNs (C-CNN) in order to solve this problem [19]. The advantage of using multi-objective CNN is the ability to train the network on multiple tasks simultaneously while re-using the same features instead of having to re-learn some of them when using C-CNNs. Similar multi-objective CNN approaches have been used for detecting facial landmarks, face recognition and localisation as well as orientation [20] [21]. Other similar approaches can be done to optimise the training of the network on two GPUs, each one following each branch [22]. Also, mid-layer parameter transfer between two identical networks, but each one having different sets of objective labels has proven to be effective [23].

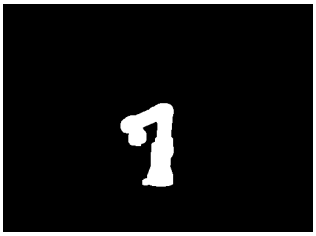
This paper is organized as follows. We present the system setup and dataset collection in Section II. Then, we explain the proposed method and CNN architecture in Section III and the training procedure in Section IV. We provide experimental results in Section V, followed by relevant conclusions and future work in Section VI.

II. SYSTEM SETUP AND DATASET COLLECTION

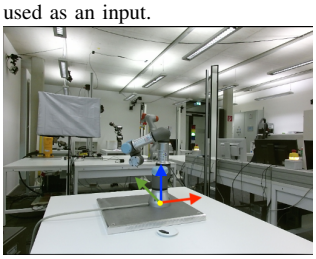
Deep learning typically requires large amounts of diverse training data for robust learning. However, this is an issue for industrial robotics applications, because there are close to none existing public datasets with well-marked ground truth data. Thus, in order to get reliable training data, a new



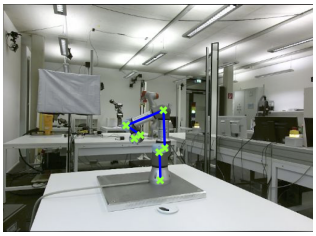
(a) Color image from the dataset used as an input.



(b) Ground truth model of the robot mask.



(c) Ground truth data of the robot base 3D position in relation to the camera marked on the input image.



(d) Ground truth data of the 3D position of robot joints marked on the input image.

Fig. 2. Example image of the dataset and ground truth examples of the UR3 robot.

dataset was created specifically for the presented application. The whole range of Universal Robots: UR3, UR5 and UR10, were used at three institutions: TU Graz, Joanneum Research and the University of Oslo. All three robots share similar visual appearance, but differ significantly in size, reach and payload capabilities.

As a vision sensor, a Kinect V2 camera is used [24]. It provides both color image and depth information. Depth images are only used for the creation of the ground truth data, while the whole following recognition process is using just a color image as an input.

For each recording, in order to have a precise ground truth data, Kinect was placed at arbitrary position observing the workspace of the robot. At each position, a Hand-Eye calibration was performed by placing a marker on the end-effector of the robot and using both color and depth image for the calibration process [25]. This provides an accurate coordinate frame transformation between the camera and the base of the robot, with an error below 0.52 cm for all the datasets.

TABLE I. Dataset summary describing a number of samples collected for each type of the robot. In total 9 recordings were made, 3 for each type of robot.

Recording	Robot Type	Number of Samples
Rec 1	UR3	211
Rec 2	UR3	252
Rec 3	UR3	463
Rec 4	UR5	252
Rec 5	UR5	756
Rec 6	UR5	1512
Rec 7	UR10	112
Rec 8	UR10	278
Rec 9	UR10	514

Once the transformation is known, a mask defining the location of the robot in the camera image can be calculated. It is done by utilising the encoder information from each joint of the robot and using a simplified model of the robot. The robot is represented using basic cylindrical and spherical shapes in 3D space according to its model and then mapped onto a virtual 2D image representing the sight of the camera. Thresholding this image results in a robot body mask representation in the camera image. The MoveIt! package was used to implement this method [26].

The robot should be observed from all the different angles and in a high variety of joint angle configurations to achieve good robustness. Movements for the data collection were programmed to provide a high diversity of viewpoints and robot body configurations. Each robot joint is moved through the full range of motion in combination with other joints as well. The step size of the joint movements is varied between the datasets resulting in a different number of samples in each. After each movement, a trigger signal is used to save the data. At each instance, camera images, joint coordinates, Cartesian coordinates of each joint and ground-truth robot mask images were saved. The number of samples per dataset varied from 112 to 1512. The variation was caused by different types and resolutions of programmed robot movements during the data collection. In total 9 datasets were collected, 3 for each type of the robot, summarized in Table I. Example images from the collected dataset are shown in Figure 1. Datasets with UR5 robot were the most extensive given the access to the robot at the lab of the main author. An example of color and ground truth of robot mask, base position and joint positions can be seen in Figure 2.

Recorded images have 512×424 pixel resolution and they are all rectified to compensate for lens distortion. Internal camera calibration was used to ensure that both color and depth information have a good overlap, avoiding any offsets. Random sampling was used to divide the final dataset into the training set and the test set by ratios of 80% and 20% of all the images respectively.

III. METHOD

Our approach is based on a multi-objective CNN structure. This approach allows us to get multiple outputs of different types by having just one input. It is achieved by having a number of convolutional layers, which are common for the whole system and then branching out the structure for each of the objectives. The whole system is trained simultaneously, meaning that the features in common layers are reused.

In our case, we train for four objectives:

- Robot mask in the image
- Robot type
- 3D Robot base position in relation to the camera
- 3D Position of the robot joints

The structure of the CNN is shown in Figure 3. It consists of the two main branches. The first one learns a classification task of finding the robot in the input image. It results in a robot mask defining the location of the robot. The second branch is for the regression tasks of finding the 3D robot base coordinates in relation to the camera and the 3D coordinates of each of the robot joints. In addition, on the same branch, the classification of the robot type is done.

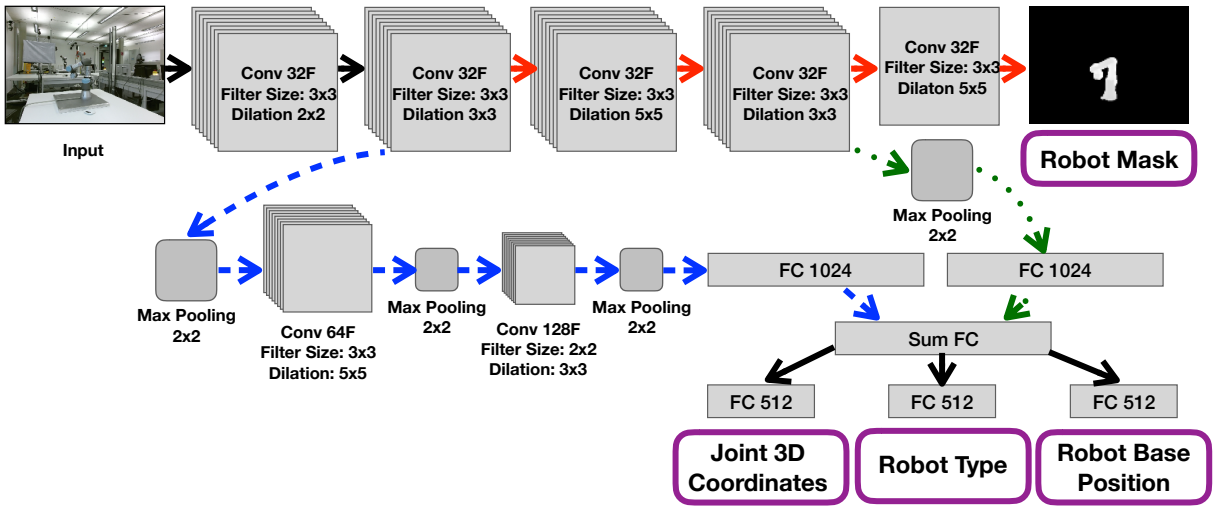


Fig. 3. Multi-objective CNN structure. Input is a simple 2D color image and the network is trained for four outputs: robot mask, 3D coordinates of robot joints, 3D coordinates of robot base position and robot type. There are two main branches of the CNN. The first one is aimed to learn the features leading to an accurate robot mask mainly consisting of dilated convolutional layers. It is marked by red solid arrows. The second branch, marked in blue dashed arrows, consists of a number of max pooling and dilated convolutional layers with fully connected layers at the end. The goal is to predict coordinates as a regression task as well as classify the robot type. Additionally, there is a branch starting from the 4th convolutional layer of robot mask task to the end of the blue branch using summing of fully connected layers, marked in dotted green arrows. It adds the information of features well defining the visual representation of the robot to the other tasks further improving the results. The whole CNN is trained for all four outputs simultaneously using a common loss function.

In addition, there is the second branch from the 4th convolutional layer towards the robot mask, which connects to the second branch. Given the idea that robot body parts are learned quite well for the robot mask classification task, this additional input provides the essential information for identifying the location of the robot joints. Fully connected layers, which are summed, are believed to filter the important visual cues and assist for the coordinate regression tasks.

A. Loss Functions

Loss functions are used to determine the quality of training. Given we have four objectives, we first describe loss functions for each one. Because the network is trained for all of the objectives simultaneously, finally we combine all four loss function into one used for the actual training.

The loss function for the robot mask was designed to adjust for a small area the foreground object takes up in the input image. In our datasets, the area taken up by the robot body in the input image was varying between $6-17\%$ of the whole image. If the loss function does not compensate for this, the CNN could classify all the pixels as background and still achieve the accuracy of 83 to 94%, which is conceptually wrong. To prevent this, the foreground weight w_{fg} is calculated, as described in Equation 1. It is based on the inverse probability of the foreground and background classes, where $Y \in \{fg, bg\}$.

$$w_{fg} = \frac{1}{\mathsf{P}(Y = fg)} \tag{1}$$

The background weight w_{bg} is calculated in Equation 2.

$$w_{bg} = \frac{1}{\mathsf{P}(Y = bg)} \tag{2}$$

The robot mask loss function is calculated in two steps. First, a per-pixel loss l^n is calculated in Equation 3, where i_{est} is $\mathsf{P}(Y = fg)$, $(1 - i_{est})$ is $\mathsf{P}(Y = bg)$ and i_{gt} is the ground truth value from the mask image.

$$\begin{aligned} l^n(I^n_{est}, I^n_{gt}) = &- w_{fg} i_{est} \log(i_{gt}) \\ &- w_{bg}(1 - i_{est}) \log(1 - i_{gt}) \end{aligned} \tag{3}$$

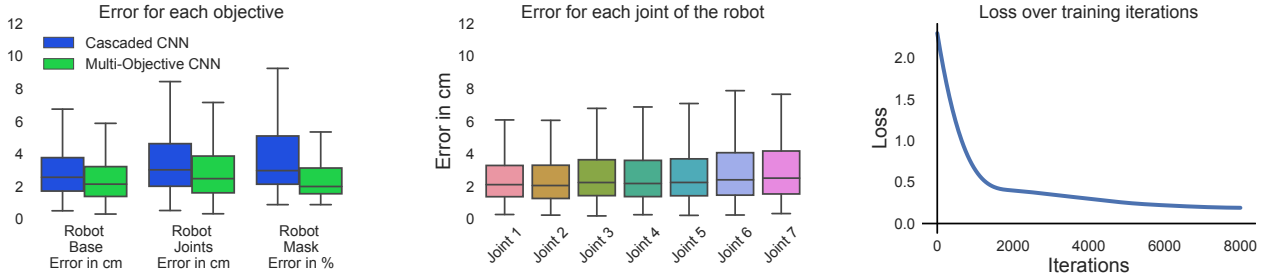
Then, it is used as an input to calculate normalised loss for the whole image $\mathcal{L}_{mask}$ in Equation 4. A normalisation factor $\mathcal{N}$, which is a number of pixels in the image, allows us to keep the learning rate fixed, despite the variance of the input image size.

$$\mathcal{L}_{mask}(I_{est}, I_{gt}) = \frac{1}{\mathcal{N}} \sum_n l^n(i_{est}, i_{gt}) \tag{4}$$

Loss functions for both robot base coordinates and the coordinates of the robot joints are formulated as regression tasks. Both of them use Euclidean distance between estimated and ground truth values. Loss function for the 3D coordinates of robot joints $\mathcal{L}_{Jcoords}$ is described in Equation 5, where N_j is the number of joints, J_i defines ground truth position of each joint and E_i is the estimated position of each joint by the CNN.

$$\mathcal{L}_{Jcoords} = \frac{1}{N_j} \sum_{i=1}^{N_j} \|J_i - E_i\|_2 \tag{5}$$

Similarly, the loss function for the coordinates of the robot base $\mathcal{L}_{Bcoords}$ is shown in Equation 6. B_{xyz} is the ground truth position of the robot base in 3D and E_{xyz} is the estimated 3D position of the robot base. These positions are relative to the camera. Considering the goal of detecting the position of the robotic manipulator, estimating just Cartesian



(a) Comparison of the Multi-Objective CNN approach against the Cascaded CNN approach [19]. Errors for all the objectives are smaller, as well as the range of quartile values.

(b) Error for the 3D coordinate estimation for positions of each robot joint. It can be seen that the error slightly increases for joints further away from the base.

(c) Value of the loss function on the test set over iterations during the training process.

Fig. 4. Evaluation of our method by testing the trained system on the test dataset.

coordinates is sufficient. If necessary, the angles of each joint in relation to the robot base could be calculated by using coordinate positions.

$$\mathcal{L}_{Bcoords} = \left\| B_{xyz} - E_{xyz} \right\|_2 \tag{6}$$

The loss function to identify the robot type was defined as a categorical cross-entropy problem. It is commonly used for multi-class classification problems. $\mathcal{L}_{type}$ is calculated in Equation 7, where p is the ground truth labels, q are the predicted labels and $c \in R$, where R are all the available types of robots in the dataset.

$$\mathcal{L}_{type} = -\sum_{c} p(c) \log q(c) \tag{7}$$

The final loss function $\mathcal{L}_{final}$ is a weighted combination of all four previously described functions. The larger the weight W, the higher the emphasis on the correct prediction of the corresponding value. And the weights should be selected to have a good overall performance of the system. The calculation of $\mathcal{L}_{final}$ is described in Equation 8.

$$\begin{aligned} \mathcal{L}_{final} = & W_{mask}\mathcal{L}_{mask} + W_{Jcoords}\mathcal{L}_{Jcoords} \\ & + W_{Bcoords}\mathcal{L}_{Bcoords} + W_{type}\mathcal{L}_{type} \end{aligned} \tag{8}$$

In order to keep the CNN easily adaptable to other types of robots in the future, no prior information about the robot model is incorporated in the system. The raw CNN output is used to evaluate the accuracy of the results.

IV. CNN TRAINING

Training of the multi-objective CNN is done for all four objectives at the same time. One possibility to adjust the quality of results is to adjust the weights given to the loss functions of each of the objectives when defining the final loss function of the system. In our case, the weight values were hand-selected using trial and error during the testing phase. Selected weight values were the following:

- W_{mask}: 1.0
- $W_{Jcoords}$: 1.5
- $W_{Bcoords}$: 1.5
- W_{type}: 0.3

The training is done on the training set, including images of all three types of robots simultaneously. In total 926 samples for UR3, 2520 samples for UR5 and 904 samples for UR10 were used

In order to speed up the process and have reasonably sized mini-batches, the input size of the images was reduced by half from the original dimensions, down to 256×212 pixels. The pixel intensity values of the input images were normalised to the range between 0 and 1. Furthermore, pixel values of the ground truth images are clipped to avoid division by zero in cases when the estimated mask fits the ground truth perfectly. In order to avoid any training biases, the data were randomly shuffled and split into mini-batches of 64 images each, fully utilising the memory of the GPU. The learning rate was set to 0.001 at the beginning of the training and then gradually decreased to 0.000001 as the training progressed. The CNN converged after 8000 iterations. It took 60 hours to train the system using a regular NVIDIA GeForce 1080 GTX graphics card.

V. RESULTS

The evaluation was done by testing our network on the test set and comparing the output against the ground truth data. The robot mask accuracy is defined by comparing a number of pixels in the CNN output image that match the ground truth mask. For the robot joint and base coordinates, Euclidean distance between the CNN estimated results and ground truth results was calculated. Robot type accuracy was computed by counting the percentage of correct classification instances. We compare the results against our previously presented C-CNN approach [19].

Robot mask classification achieved an accuracy of 98%, which is almost 3% improvement compared to our previous method, as seen in Figure 4(a). A significant amount of this error comes from failing to estimate sharp corners in the mask image because CNN outputs slightly blurry mask compared to ground truth. It is likely that some post-processing would allow even further improvement by increasing the sharpness of the mask.

The overall error of the 3D position of robot joints was 3.16 cm, which is a slight improvement compared to the error of 3.32 cm in our previous work. If we analyse each

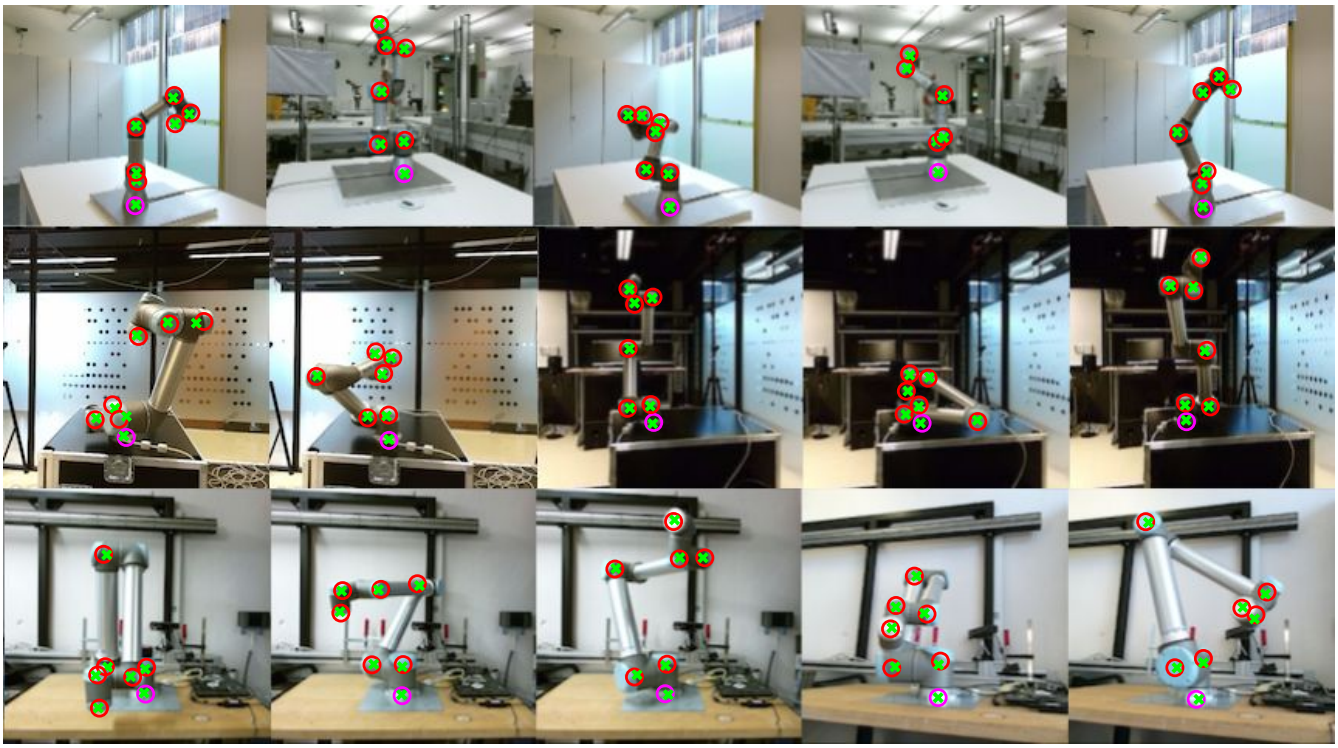


Fig. 5. Estimated robot joint position coordinates marked on the images taken from the dataset. Due to difficulty in visualising 3D coordinates on printed figures, the estimated joint coordinates were mapped back into 2D images. Green crosses indicate the ground truth position, red circles indicate predicted positions of joints and magenta circles indicate the predicted position for the robot base.

joint separately, we can see the tendency of the joints closest to the robot base having a smaller average error, as well as smaller scatter, compared to the joints closer to the end-effector. Results are showing that in Figure 4(b). It can be explained by analysing the reachability from the base of each of the joint. The end effector has the largest range of motion, and it reduces for the joints closer to the robot base. This means the range of possible positions varies significantly, and estimation is more difficult in the larger range of possible positions. However, the error difference is minor.

TABLE II. Summary of the results on the test set of a Multi-Objective CNN with a comparison to our previous work using C-CNN.

Measure	Current Work	Previous Work
Mask Accuracy, %	**98%**	94.6%
Robot Type Accuracy, %	**98.3%**	—
Joint Pos Error (Mean)	$\mathbf{3.16cm}$	$3.32cm$
Base Pos Error (Mean)	$\mathbf{2.74cm}$	$2.97cm$

The estimation of the position of the robot base in relation to the camera had an average error of 2.74 cm. Once again, this is lower compared to C-CNN approach, where the same estimation error was 2.97 cm. Robot type classification made just a few wrong decisions resulting in 98.3% accuracy. The forward propagation time (detection speed) of the neural network was on average 15 ms for one image, making it suitable for real-time applications.

The final results are summarised in Table II, and the estimated coordinates by the full system marked over the dataset images can be seen in Figure 5. Because it is difficult to show 3D estimations on 2D figures, the visualisation of estimation is done by mapping the estimated 3D coordinates back onto input images.

Both in the current multi-objective CNN approach and the C-CNN method, we used exactly the same datasets for training and testing, so the results can be compared directly. Given the lack of similar work, no suitable benchmark was found to allowing a direct comparison of achieved results.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a solution for detecting a robot manipulator and estimating the positions of its joints in a 2D camera image. A camera can be placed in arbitrary positions overlooking the robot workspace and the method successfully localizes the robot without the need for any additional setup or Hand-Eye calibration. This provides more flexible and quickly reconfigurable environment aware robotic setups for tasks like human-robot or robot-robot interaction. We have used three types of robots produced by Universal Robot for training and testing of the system: UR3, UR5 and UR10.

Our system uses a multi-objective convolutional neural network approach to achieve the goal. It optimises the system for four objectives simultaneously provides the mask of an area where the robot is present in the camera image, its base position in relation to the camera, 3D positions of the joints of the robot as well as the type of the robot, respectively the 3D joint position error was less than 3.16 cm, the robot mask accuracy was 98% and the robot type was successfully recognised in over 98.3% of cases. These results are an improvement of our previously presented C-CNN approach, both in accuracy and flexibility of the system.

Given current results, the continuation of work will be to apply this method in more complex environments containing multiple robots and people working in the same workspace. Self-occlusions were present in the tested datasets and some minor occlusions of other objects, however, more evaluation is needed using cases like people or other machinery passing by between the camera and the robot blocking the view.

This work has multiple possible applications. One would be the safety aspect of identifying robots in robotised environments like factory floors, warehouses or automated surgery rooms where an operator has a wearable camera detecting robots in the field of view. Another application would be for robot-robot interaction. With swarm robotics, both homogeneous and heterogeneous, and different sizes, direct communication between them is not always reliable. Our approach would allow the robots to observe and track each other using small cameras and identify the intentions of other robots in the surroundings.

For the human-robot collaboration tasks, a person tracking can be achieved using devices like Leap Motion or skeleton tracking to estimate of the relative hand positions to the robot. This can be used for tool handover between the person and the robot, working towards a common goal or even hand-gesture control, while avoiding any unwanted physical contact between the two.

In the future, we plan to test the system with more types of the robots by using transfer learning on pre-trained CNN. In this case, the dataset needed to teach to identify a new robot type should be significantly reduced compared to the current setup. Adding human skeleton tracking would move the work closer to the real-world human-robot interaction tasks. The system will be tested in some use case scenarios to identify the robustness in less controlled environments with more illumination changes and changing setups.

ACKNOWLEDGMENT

This work is partially supported by The Research Council of Norway as a part of the Engineering Predictability with Embodied Cognition (EPEC) project, under grant agreement 240862, and by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) within the project framework CollRob (Collaborative Robotics).

REFERENCES

[1] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[2] J. Roach and M. Boaz, "Coordinating the motions of robot arms in a common workspace," *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 437–444, 1987.

[3] J. Leitner, S. Harding, M. Frank, A. Förster, and J. Schmidhuber, "Transferring spatial perception between robots operating in a shared workspace," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1507–1512.

[4] C. Fitzgerald, "Developing Baxter," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–6.

[5] J. Miseikis, K. Glette, O. J. Elle, and J. Torresen, "Automatic calibration of a robot manipulator and multi 3d camera system," in *System Integration (SII), 2016 IEEE/SICE International Symposium on*. IEEE, 2016, pp. 735–741.

[6] S. Schneegans, P. Vorst, and A. Zell, "Using RFID Snapshots for Mobile Robot Self-Localization." in *EMCR*, 2007.

[7] J.-S. Gutmann and C. Schlegel, "Amos: Comparison of scan matching approaches for self-localization in indoor environments," in *Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on*. IEEE, 1996, pp. 61–67.

[8] O. Stasse, A. Escande, N. Mansard, S. Miossec, P. Evrard, and A. Kheddar, "Real-time (self)-collision avoidance task on a hrp-2 humanoid robot," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 3200–3205.

[9] A. De Santis, A. Albu-Schaffer, C. Ott, B. Siciliano, and G. Hirzinger, "The skeleton algorithm for self-collision avoidance of a humanoid manipulator," in *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*. IEEE, 2007, pp. 1–6.

[10] W. J. Wilson, C. W. Hulls, and G. S. Bell, "Relative end-effector control using cartesian position based visual servoing," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 684–696, 1996.

[11] A. Ruf, M. Tonko, R. Horaud, and H.-H. Nagel, "Visual tracking of an end-effector by adaptive kinematic prediction," in *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, vol. 2. IEEE, 1997, pp. 893–899.

[12] B. Daachi and A. Benallegue, "A neural network adaptive controller for end-effector tracking of redundant robot manipulators," *Journal of Intelligent & Robotic Systems*, vol. 46, no. 3, pp. 245–262, 2006.

[13] I. Siradjuddin, L. Behera, T. M. McGinnity, and S. Coleman, "Image-based visual servoing of a 7-DOF robot manipulator using an adaptive distributed fuzzy PD controller," *IEEE/ASME Transactions on Mechatronics*, vol. 19, no. 2, pp. 512–523, 2014.

[14] M. R. Soltanpour and M. H. Khooban, "A particle swarm optimization approach for fuzzy sliding mode control for tracking the robot manipulator," *Nonlinear Dynamics*, vol. 74, no. 1-2, pp. 467–478, 2013.

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

[16] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3389–3396.

[17] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[18] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[19] J. Miseikis, P. Knobelreiter, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, "Robot Localisation and 3D Position Estimation Using a Free-Moving Camera and Cascaded Convolutional Neural Networks," *ArXiv e-prints*, Jan. 2018.

[20] X. Yin and X. Liu, "Multi-task convolutional neural network for pose-invariant face recognition," *IEEE Transactions on Image Processing*, 2017.

[21] Y. Wu, T. Hassner, K. Kim, G. Medioni, and P. Natarajan, "Facial landmark detection with tweaked convolutional neural networks," *arXiv preprint arXiv:1511.04031*, 2015.

[22] 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.

[23] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[24] P. Fankhauser, M. Bloesch, D. Rodriguez, , R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *IEEE International Conference on Advanced Robotics (ICAR) (submitted)*, 2015.

[25] T. Heikkilä, M. Sallinen, T. Matsushita, and F. Tomita, "Flexible hand-eye calibration for multi-camera systems," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2000, pp. 2292–2297.

[26] I. A. Sucan and S. Chitta, "MoveIt!" *Online Available: http://moveit.ros.org*, 2013.

PAPER VI

Transfer Learning for Unseen Robot Detection and Joint Estimation on a Multi-Objective Convolutional Neural Network

J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen

2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR), Shenyang, 2018, pp. 337-342.

Winner of the Best Student Paper Award

VI

Transfer Learning for Unseen Robot Detection and Joint Estimation on a Multi-Objective Convolutional Neural Network

Justinas Mišeikis[1], Inka Brijacak[2], Saeed Yahyanejad[3], Kyrre Glette[4], Ole Jakob Elle[5], Jim Torresen[6]

***Abstract*— A significant problem of using deep learning techniques is the limited amount of data available for training. There are some datasets available for the popular problems like item recognition and classification or self-driving cars, however, it is very limited for the industrial robotics field. In previous work, we have trained a multi-objective Convolutional Neural Network (CNN) to identify the robot body in the image and estimate 3D positions of the joints by using just a 2D image, but it was limited to a range of robots produced by Universal Robots (UR). In this work, we extend our method to work with a new robot arm - Kuka LBR iiwa, which has a significantly different appearance and an additional joint. However, instead of collecting large datasets once again, we collect a number of smaller datasets containing a few hundred frames each and use transfer learning techniques on the CNN trained on UR robots to adapt it to a new robot having different shapes and visual features. We have proven that transfer learning is not only applicable in this field, but it requires smaller well-prepared training datasets, trains significantly faster and reaches similar accuracy compared to the original method, even improving it on some aspects.**

I. INTRODUCTION

Industrial robotics has been associated with structured and well-defined environments for many years and robot arms have achieved great performance in areas like manufacturing. It comprises of hard-coded repetitive motions, where a machine can do a better job compared to a person in terms of no fatigue, precision and non-stop operation. However, with developing hardware, computing power and advancing algorithms, the same systems are becoming more adaptive. Nowadays, instead of fencing off the robots, environment understanding and adaptive behaviour is a part of the Industry 4.0 concept, where robots and people can share the same workspace and collaborate [1].

There are numerous approaches to sense the environment: laser scanners, stereo vision, RGB-D cameras, camera arrays, ultrasound sensors, motion capture systems. Each one has its own pros and cons, often either needing additional markers or calibrated devices or having a high price-tag. Very often there is still a significant amount of work needed to set up a new robustly working system.

Inspiration of the environment understanding comes from biology - how animals and especially humans are able to understand the environment. We are capable of learning what objects are, how they move, their functionality and the way we should interact with them by looking at example situations. Furthermore, after we know how it works in some situation, it is very likely that next time we see similar conditions, we will be able to find parallels between the two and figure out how we should act by simply using our previously gained knowledge. That is the motivation of the transfer learning method, which uses a previous well-trained neural network and adjusts it to new conditions using limited amount of training data and significantly shorter training time compared to the full training of the neural network.

Transfer learning has been used in a variety of fields. In many cases, the whole or part of the CNN trained on ImageNet is taken as a base network and then adjusted to a specific application [2]. This has been proven to work for mid-level image representations in object classification, using the pre-trained network on natural images to adapt for medical image recognition and even emotion recognition [3] [4] [5]. Another interesting application of transfer learning is to use a fully trained network on night-time satellite imagery of poverty areas and adapt it to recognise poverty areas from daytime satellite imagery [6]. Furthermore, detailed analyses of the transfer learning approaches were made with surveys of the techniques used and various CNN structures [7] [8].

The proof that generalised visual features can be transferred to new systems has motivated to use it to extend our previous work of recognising the robot and estimating its 3D position of the joints by using a simple 2D color camera image [9]. Instead of using ImageNet or any other well known pre-trained network, we take our previously fully trained multi-objective CNN on Universal Robots and use it to adapt to a new Kuka LBR iiwa robot arm. Additionally, the new dataset adds new unseen backgrounds making the network even more robust.

The main goal of identifying the robot in a 2D camera image is to remove the need for fully calibrated camera-robot systems allowing for more dynamic environments, while still ensuring safe operations. It is crucial for shared workspaces between humans and robots. There are many good methods of real-time dynamic obstacle and people avoidance, but most of them require a fully-calibrated robot-camera system [10] [11]. Despite some efficient Hand-Eye calibration methods, it is still a cumbersome process when the operation of the robot has to be halted until the calibration is completed [12]. Furthermore, it can simplify the task of having mobile robots moving around the floor without any special markings. By identifying other fixed robots it

[1 4 5 6]Justinas Mišeikis, Kyrre Glette, Ole Jakob Elle and Jim Torresen are with the Department of Informatics, University of Oslo, Oslo, Norway

[2 3] Inka Brijacak and Saeed Yahyanejad are with the Joanneum Research - Robotics, Klagenfurt am Wörthersee, Austria

[5]Ole Jakob Elle has his main affiliation with The Intervention Centre, Oslo University Hospital, Oslo, Norway oelle@ous-hf.no

[1 4 6] {justinm,kyrrehg,jimtoer}@ifi.uio.no

[2] Inka.Brijacak@joanneum.at

[3] Saeed.Yahyanejad@joanneum.at

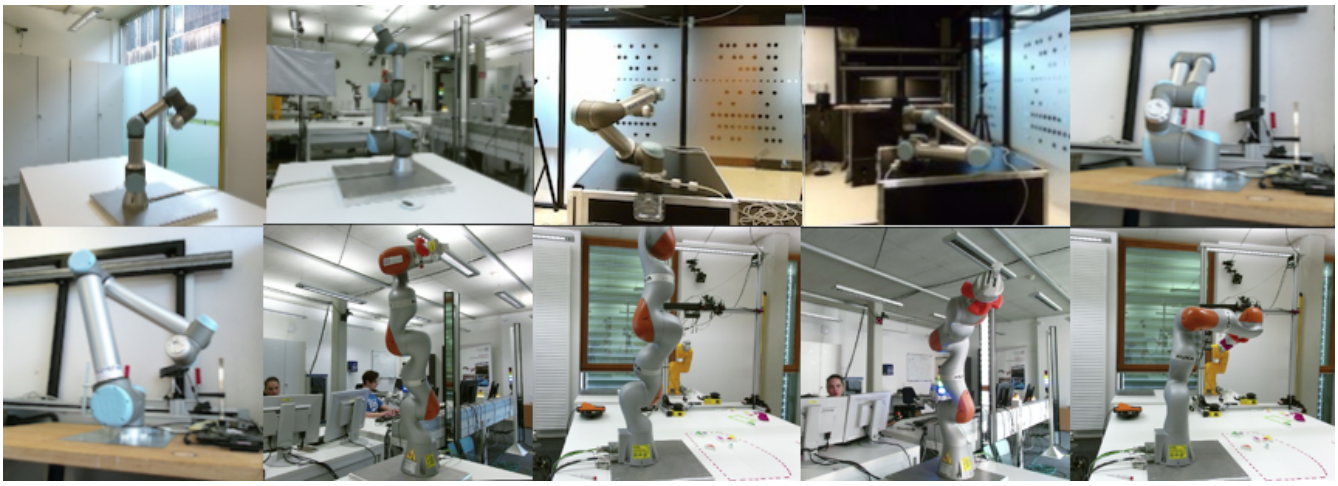


Fig. 1. Samples from the collected robot datasets. It consists of a line of Universal Robots (silver-blue) as well as Kuka LBR iiwa (silver-orange). The data was collected with a variety of backgrounds and light conditions to provide more robustness.

can both avoid possible collisions and localise itself to known fixed-base robots. By identifying other robots and knowing their exact position, the setup could be expanded to prediction of the behaviour of other machinery in the surrounding environment without having the direct communication channel between them. This would be a very useful approach in swarm robotic applications.

This paper is organized as follows. We present the system setup and dataset collection in Section II. Then, we explain the proposed method and CNN structure and configuration in Section III and the transfer learning procedure in Section IV. We provide experiments and results in Section V, followed by relevant conclusions and future work in Section VI.

II. SYSTEM SETUP AND DATASET COLLECTION

Training a deep learning network typically requires a large amount of diverse training data. The main problem lies in the necessity to have precise ground-truth information, which is given as a correct answer.

Our setup consisted of a vision sensor, in this case, a Kinect V2 camera, placed in arbitrary positions overlooking the robot and perform Hand-Eye calibration at each of the positions [13]. The calibration is done by placing a known marker on the end-effector of the robot and performing a number of movements until the calibration accuracy reaches the necessary precision. The result is a coordinate frame transformation between the camera and the robot base [14].

Given a precise coordinate frame transformation, the robot model is used together with the live information from its joint encoder readings to create a simplified mesh model defining the robot shape. Then it is transformed to the coordinate frame of the camera and depth image estimated from the viewpoints of the camera. The result is a precise mask of the robot body in the camera image, which can be overlayed with a color image and used as a ground truth data for teaching the CNN. The main benefit is that this process is fully automated by using ROS with MoveIt! package [15]. The robot model is taken from the Unified Robot Description Format (URDF) files provided by the robot manufacturers [16].

In our experiments, we use an already trained multi-objective CNN from the previous project [9], which was trained from scratch on three robot models from Universal Robots: UR3, UR5 and UR10. In order to test the capabilities of transfer learning, new datasets using Kuka LBR iiwa were used. For comparison reasons, relatively large datasets, summarised in Table I, were collected for all the robots. These datasets consist of multiple recordings, each one with the camera placed at different angles and distances relative to the robot as well as having various backgrounds.

TABLE I. Dataset summary describing a number of samples collected for each type of the robot.

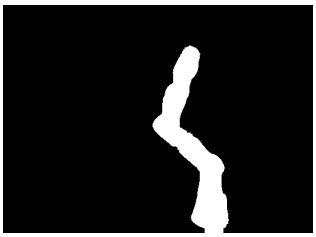
Robot Type	Number of Datasets	Total Number of Samples
Universal Robots	9	4350
Kuka LBR iiwa	14	1837

Robot movements included a large variety of joint configurations resulting in many viewpoints of the robot. Furthermore, lighting conditions were varied for each of the recordings to allow for more robustness regarding the brightness and reflections.

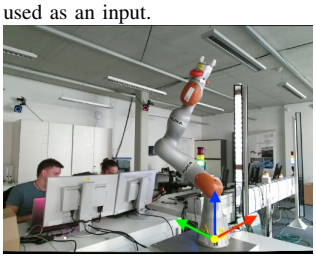
The new datasets with the Kuka robot also included more dynamic background with people moving around and even another Kuka robot placed further away and not being used in experiments. Furthermore, in some cases, the robot went out of bounds of the color image. In total, 9 datasets of Universal Robots and 14 datasets of Kuka robot were used. Each recording had different camera placement, changing distance between the robot and the camera, varying lighting conditions and new background. During each of the recordings, the robot was moving to give a large variety of joint configurations in the dataset.

At the completion of each movement, a trigger signal was sent in order to save the color image, depth model, cartesian and joint coordinates of each of the robot joint and ground-truth mask model of the robot. All this information was later used to train the neural network. However, depth information was used only for training, while the recognition part of the

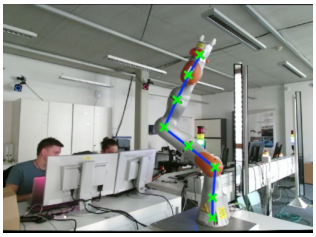
(a) Color image from the dataset used as an input.



(b) Ground truth model of the robot mask.



(c) Ground truth data of the robot base 3D position.



(d) Ground truth data of the 3D position of robot joints.

Fig. 2. Dataset, mask and ground truth value examples of the Kuka LBR iiwa robot.

system relies only on the color camera image as an input.

In order to normalise the input data, internal camera calibration was used to ensure a perfect overlap between color and depth images. All the input images are also rectified and have the resolution of 512×424 pixels. Testing and validation sets were divided by the ratios of 80% and 20% respectively based on random sampling.

III. CNN STRUCTURE AND CONFIGURATION

The base of a multi-objective CNN is taken from previous work, where it was trained on a line of robots made by Universal Robot [9]. The network simultaneously optimises for multiple heterogeneous outputs by using just a single image as an input.

The network in this paper is trained on four objectives:

- Robot mask in the image
- Robot type
- 3D Robot base position in relation to the camera
- 3D Position of the robot joints

The structure of the CNN is shown in Figure 3. The network shares a number of common convolutional layers and then branches for more objective-specific optimisation. Having a single training process, it means that the features in common layers are reused.

A. Loss Functions

Loss functions are used to evaluate the training progress and the achieved accuracy compared to the ground truth data. Our system optimises for four objectives simultaneously, resulting in four loss functions, which are later combined into one for the training process. First, each of the loss functions will be described separately followed by the explanation of how they are all connected into one.

The robot body takes up a relatively small area in the whole image. The area taken up by the robot body in UR datasets is between $6 - 17\%$ and for Kuka datasets, it is between $8 - 18\%$ of the whole image. Given a standard pixel classification loss function, there would be cases when an accuracy of over 82% can be reached by classifying the whole image as a background. That is conceptually wrong, so the loss function was adjusted by using the foreground weight w_{fg}, which is calculated in Equation 1. It is based on the inverse probability of the foreground and background classes, where $Y \in \{fg, bg\}$.

$$w_{fg} = \frac{1}{\mathsf{P}(Y = fg)} \tag{1}$$

The background weight w_{bg} is calculated in Equation 2.

$$w_{bg} = \frac{1}{\mathsf{P}(Y = bg)} \tag{2}$$

The loss function for the robot mask is defined by two steps. First, a per-pixel loss l^n is calculated in Equation 3, where i_{est} is $\mathsf{P}(Y = fg)$, $(1 - i_{est})$ is $\mathsf{P}(Y = bg)$ and i_{gt} is the ground truth value from the mask image.

$$\begin{aligned} l^n(I^n_{est}, I^n_{gt}) = &- w_{fg} i_{est} \log(i_{gt}) \\ &- w_{bg}(1 - i_{est}) \log(1 - i_{gt}) \end{aligned} \tag{3}$$

This is followed by a normalised loss calculation for the whole image $\mathcal{L}_{mask}$ in Equation 4. A normalisation factor $\mathcal{N}$, which is the number of pixels in the image, allows us to keep the same learning parameters independent of the input image size.

$$\mathcal{L}_{mask}(I_{est}, I_{gt}) = \frac{1}{\mathcal{N}} \sum_n l^n(i_{est}, i_{gt}) \tag{4}$$

3D coordinates of the robot base and robot joints are defined as regression tasks. The loss function is based on the Euclidean distance between the estimated values and the ground truth values. For the robot joints estimation, the loss function $\mathcal{L}_{Jcoords}$ is described in Equation 5, where N_j is the number of joints, J_i is the ground truth position of each joint and E_i is the estimated values by the neural network.

$$\mathcal{L}_{Jcoords} = \frac{1}{N_j} \sum_{i=1}^{N_j} \|J_i - E_i\|_2 \tag{5}$$

The loss function for the coordinates of the robot base $\mathcal{L}_{Bcoords}$ is calculated in Equation 6. B_{xyz} is the ground truth position of the robot base in 3D, and E_{xyz} is the estimated 3D position of the robot base. These positions are relative to the coordinate frame of the camera.

$$\mathcal{L}_{Bcoords} = \left\|B_{xyz} - E_{xyz}\right\|_2 \tag{6}$$

Classification of the robot type $\mathcal{L}_{type}$ is defined as a categorical cross-entropy problem with multiple classes. $\mathcal{L}_{type}$ is calculated in Equation 7, where p is the ground truth labels, q are the predicted labels and $c \in R$, where R contains all the available types of robots in the dataset.

$$\mathcal{L}_{type} = -\sum_c p(c) \log q(c) \tag{7}$$

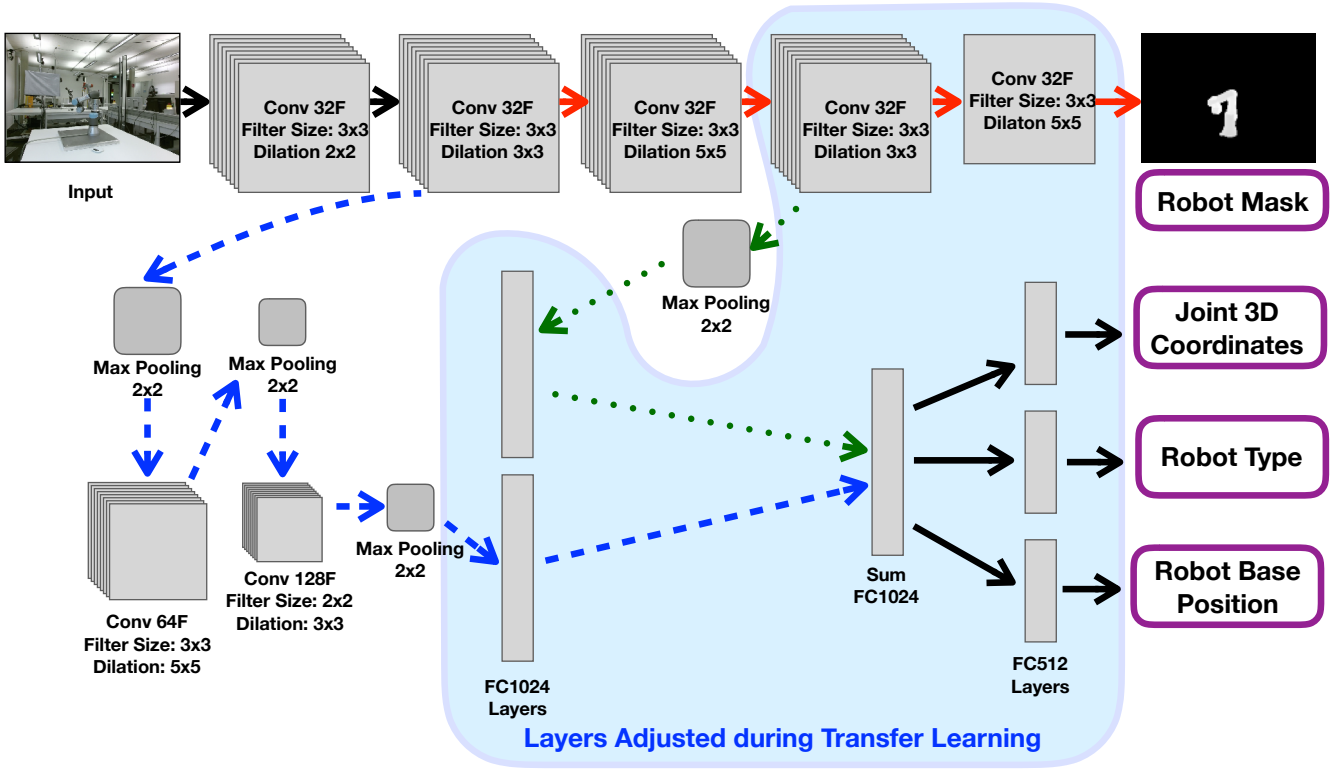


Fig. 3. Structure of the multi-objective CNN. Input is a 2D color image resulting in four outputs: robot mask, 3D coordinates of robot joints, 3D coordinates of the robot base and the robot type. The network uses part of common convolutional layers and then branches off for objective-specific training. Fully-connected (FC) layers, marked by blue area, are the ones being adjusted during the transfer learning process to adapt to the new robot model. Convolutional layers learn generalised visual features of the image, so their parameters stay *frozen* during the transfer learning. This allows for quicker adaptation with a limited number of input images compared to the full training process. The whole CNN is trained for all four outputs simultaneously using a common loss function. Differently colored arrows represent connections of different branches in multi-objective CNN, each one focused for a certain type of output.

For the training of the multi-objective CNN and optimisation for all four objectives, a single loss function is needed. This was achieved by combining the previously defined loss functions into $\mathcal{L}_{final}$ by having a weight element for each of the losses, as shown in Equation 8. The larger the weight W, the higher the impact on the corresponding value.

$$\begin{aligned}\mathcal{L}_{final} = & W_{mask}\mathcal{L}_{mask} + W_{Jcoords}\mathcal{L}_{Jcoords} \\ & + W_{Bcoords}\mathcal{L}_{Bcoords} + W_{type}\mathcal{L}_{type}\end{aligned} \tag{8}$$

IV. TRANSFER LEARNING AND TRAINING

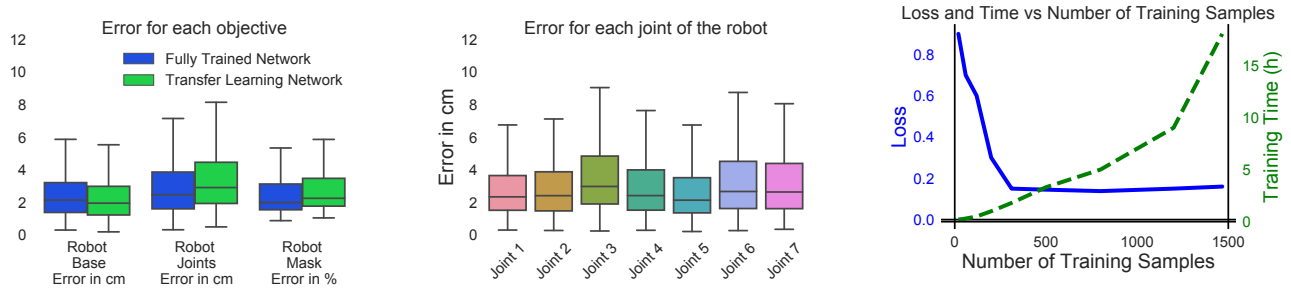
The benefit of transfer learning technique is that the parameters contained in so-called *frozen* layers are copied from the previously trained network, while only part of layers is trained during the process. This speeds up the training process and requires smaller training datasets compared to the full CNN training. In this work, most of the convolutional layers had the parameters transferred and *frozen* with all the fully connected layers and only the two last convolutional layers for robot mask estimation being trained to adapt for specific variation in visual features. They contain more robot-specific visual features, while the first layers learn more general visual features, which are more adaptable for any robot type. The exact setup is explained in Figure 3. By a layer being *frozen* it means that after the parameter transfer, they are fixed and not adjusted at all during the training.

Weights for the loss function are kept identical to the ones in previous work given good results and ability to compare the results of the works directly. Selected weight values were the following:

- W_{mask}: 1.0
- $W_{Jcoords}$: 1.5
- $W_{Bcoords}$: 1.5
- W_{type}: 0.3

One important difference between the UR robots and the Kuka robot is the number of joints. Universal Robot line has 6 joints, while Kuka has 7 joints. This difference changes the number of outputs for the 3D position estimation of robot joints. However, because the fully connected layers, as well as output layers, are trained, it can be adjusted to accommodate estimation of an extra joint.

Training was done by using datasets of different sizes containing the Kuka robot. Mini-batches were created in order to make the most out of the available GPU memory and all the data was randomly shuffled to reduce the biases. Before starting any training, parameters for the *frozen* layers were transferred from the old model fully-trained on UR datasets. This ensured that each training had an identical configuration in the beginning. The number of training samples varied by the experiment and the input size of the images was reduced by half from the original dimensions, down to 256×212 pixels. The pixel intensity values of the input images were



(a) Comparison of a fully-trained network on UR robots [9] with a transfer learning method on Kuka. Error slightly increased for joints position estimation and robot mask, however, transfer learning method was slightly more accurate in estimating the position of the robot base.

(b) Error for the 3D coordinate estimation for positions of each robot joint of the transfer learning method. It can be seen that the error slightly increases for joints further away from the base and Joint 3 has increased error compared to the surrounding joints

(c) Loss function and training time against the number of training samples used. This was acquired by running a number of experiments using input datasets of different size. Using more than 300 training samples does not give accuracy benefit, while just increasing the training time.

Fig. 4. Evaluation of the transfer learning method using the test dataset in various categories.

normalised to the range between 0 and 1. The learning rate was set to 0.001 at the start of the training and then gradually decreased towards 0.000001 as the training progressed.

V. EXPERIMENTS AND RESULTS

A number of experiments were carried out in order to determine the effectiveness of the transfer learning process. In order to find the optimum amount of training samples needed for transfer learning, each experiment consisted of a training set with different size, all randomly sampled from the Kuka dataset. The testing set was identical for all the experiments.

TABLE II. Summary of the Transfer Learning results (using 312 samples for training) on the test set of Kuka LBR iiwa robot with a comparison of a Multi-Objective CNN with just Universal Robots.

Measure	Full Training	Transfer Learning
Mask Accuracy, %	**98%**	97.3%
Robot Type Accuracy, %	**98.3%**	—
Joint Pos Error (Median)	$\mathbf{2.46cm}$	$2.87cm$
Base Pos Error (Median)	$2.13cm$	$\mathbf{2.02cm}$
Training Time (hours)	60 hours	**2** hours

The evaluation was done using a testing set by comparing the output against the ground truth data. The robot mask accuracy is defined by counting the number of pixels in the CNN output image that match the ground truth mask. For the robot joint and base coordinates, Euclidean distance between the CNN estimated results and ground truth results was calculated. We compare the results of transfer learning method trained on the Kuka robot against our previously presented multi-objective CNN fully trained for UR robots [9]. Results are summarised in Table II.

Compared to a fully trained system, the transfer learning results matched closely. As seen in Figure 4(a), the error in estimating 3D positions of robot joints was 2.87 cm compared to 2.46 cm in a fully trained system, while the robot mask accuracy difference was just 0.7% with 97.3% for transfer learning method and 98% in a fully trained CNN. Robot base position estimation was actually more accurate in transfer learning method with an error of 2.02 cm compared to 2.13 cm. Resulting positions of robot base and joints

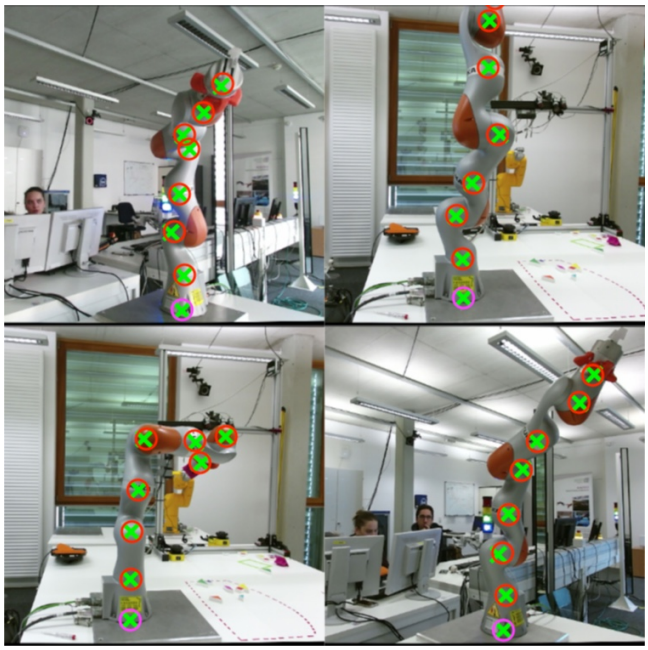


Fig. 5. Estimated robot joint position coordinates marked on the images taken from the dataset. Due to difficulty in visualising 3D coordinates on printed figures, the estimated joint coordinates were mapped back into 2D images. Green crosses indicate the ground truth position, red circles indicate predicted positions of joints and magenta circles indicate the predicted position for the robot base. In some cases, even when a part of the robot is out of bounds, positions of visible joints as well as the unseen joint are predicted with centimeter accuracy.

mapped onto 2D and marked on the example dataset images are shown in Figure 5. The system was adapted for just one type of the new robot, so we did not evaluate the accuracy of robot type detection. The system adapted to an additional robot joint in the transfer learning method. There can be seen an increase in the error for Joint 3, and that could be partly caused by a different structure of the robot, as seen in Figure 4(b).

However, the main benefit of the transfer learning method can be seen in training time. By looking at the Figure 4(c), where loss calculation and training time is shown against the number of samples used for training, it can be clearly seen that the optimum result is around 300 training samples. To be

specific, it was the experiment, where 312 training samples were used. It took a bit under 2 hours of training and the resulting loss was 0.15. Having more samples, the loss got down to 0.14, but it took significantly more time to train. An interesting point was that by using the whole training dataset, the loss increased back to 0.15 and took around 16 hours of training. The forward propagation time per sample averaged to 13.5 ms. All the training and testing was done on Nvidia Geforce GTX 1080 Ti graphics card.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a transfer learning approach to adapt a previously trained multi-objective CNN to new types of robots. In general, the system identifies and localises the robot arm and estimates its base and joints' positions in 3D. This allows a camera to be placed in any position and moved around without having to re-calibrate the camera-robot system with Hand-Eye calibration. In this work, we have shown that by taking a fully trained system, a significantly less training data is needed to adapt it for new robot models, which have a different shape, appearance and even more degrees of freedom.

The results have shown that accuracy achieved by using transfer learning closely matches the results of the fully trained system and can even improve in some cases. This means that the system is able to adapt and learn to recognise new robots with having just limited amount of training data. Similarly to what we do when learning new skills and practising them afterwards.

This work can be useful in dynamic environments where it is difficult to predict where robots, sensors and people are located, but operational safety has to be established. By expanding this method to numerous robots, and other equipment, fixed setups and calibration can be discarded. Unfortunately, the accuracy is still in centimetre level, and it is not applicable for precision tasks. However, in many adaptive human-robot and robot-robot interaction tasks, general obstacle avoidance and collaboration movements could be made possible.

Given a precise robot body detection, another possible application could be self-inspection for the robot to detect any unknown and unexpected damage. Similar to new robots are added using transfer learning, typical damages could be taught to the system and identified by the robot scanning itself, observing its own reflection or having another robot to scan it. This can be very useful in environments, like disaster areas, where robots have to work autonomously for long periods of time, or when internal sensors give unusual readings and hull should be inspected.

For future work, we plan to implement more robots as well as having robots on mobile platforms in the system. Instead of training on one new robot model, transfer learning will be used to expand the CNN to work with a line of robots, including the originally trained ones. Previously mentioned robot self-inspection is also of high interest, as well as adding collaborative tasks with people by tracking their movements using the latest skeleton tracking methods. Furthermore, more types of cameras will be tested and transition from one camera to another analysed.

ACKNOWLEDGMENT

This work is partially supported by The Research Council of Norway as a part of the Engineering Predictability with Embodied Cognition (EPEC) project, under grant agreement 240862, and by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) within the project framework CollRob (Collaborative Robotics).

REFERENCES

[1] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[2] 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.

[3] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[4] H. Greenspan, B. van Ginneken, and R. M. Summers, "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, 2016.

[5] H.-W. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler, "Deep learning for emotion recognition on small datasets using transfer learning," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*. ACM, 2015, pp. 443–449.

[6] M. Xie, N. Jean, M. Burke, D. Lobell, and S. Ermon, "Transfer learning from deep features for remote sensing and poverty mapping," *arXiv preprint arXiv:1510.00098*, 2015.

[7] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[8] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, 2016.

[9] J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, "Multi-objective convolutional neural networks for robot localisation and 3d position estimation in 2d camera images," *arXiv preprint arXiv:1804.03005*, 2018.

[10] J. Mainprice and D. Berenson, "Human-robot collaborative manipulation planning using early prediction of human motion," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 299–306.

[11] J. Mišeikis, K. Glette, O. J. Elle, and J. Torresen, "Multi 3D camera mapping for predictive and reflexive robot manipulator trajectory estimation," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–8.

[12] J. Miseikis, K. Glette, O. J. Elle, and J. Torresen, "Automatic calibration of a robot manipulator and multi 3d camera system," in *System Integration (SII), 2016 IEEE/SICE International Symposium on*. IEEE, 2016, pp. 735–741.

[13] P. Fankhauser, M. Bloesch, D. Rodriguez, , R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *IEEE International Conference on Advanced Robotics (ICAR) (submitted)*, 2015.

[14] T. Heikkilä, M. Sallinen, T. Matsushita, and F. Tomita, "Flexible hand-eye calibration for multi-camera systems," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2000, pp. 2292–2297.

[15] I. A. Sucan and S. Chitta, "MoveIt!" *Online Available: http://moveit.ros.org*, 2013.

[16] W. Meeussen, J. Hsu, and R. Diankov, "Urdf-unified robot description format," 2012.

PAPER VII

Two-Stage Transfer Learning for Heterogeneous Robot Detection and 3D Joint Position Estimation in a 2D Camera Image using CNN

J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen

2019 IEEE International Conference on Robotics and Automation (ICRA), Montreal, 2019

Two-Stage Transfer Learning for Heterogeneous Robot Detection and 3D Joint Position Estimation in a 2D Camera Image Using CNN

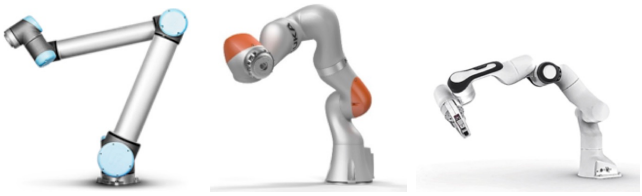
Justinas Mišeikis[1], Inka Brijačak[2], Saeed Yahyanejad[3], Kyrre Glette[4], Ole Jakob Elle[5], Jim Torresen[6]

***Abstract*— Collaborative robots are becoming more common on factory floors as well as regular environments, however, their safety still is not a fully solved issue. Collision detection does not always perform as expected and collision avoidance is still an active research area. Collision avoidance works well for fixed robot-camera setups, however, if they are shifted around, Eye-to-Hand calibration becomes invalid making it difficult to accurately run many of the existing collision avoidance algorithms. We approach the problem by presenting a stand-alone system capable of detecting the robot and estimating its position, including individual joints, by using a simple 2D colour image as an input, where no Eye-to-Hand calibration is needed. As an extension of previous work, a two-stage transfer learning approach is used to re-train a multi-objective convolutional neural network (CNN) to allow it to be used with heterogeneous robot arms. Our method is capable of detecting the robot in real-time and new robot types can be added by having significantly smaller training datasets compared to the requirements of a fully trained network. We present data collection approach, the structure of the multi-objective CNN, the two-stage transfer learning training and test results by using real robots from Universal Robots, Kuka, and Franka Emika. Eventually, we analyse possible application areas of our method together with the possible improvements.**

I. INTRODUCTION

Collaborative robots are gaining popularity as an advanced version of traditional industrial robots. Not only they are capable of reliably performing high-precision complex movements repetitively without any fatigue or rest, but they are also claimed to be safe to operate around humans. Instead of fully separating them from people (e.g., using fences or light curtains), they are capable of sharing the same workspace with humans given the sophisticated collision detection systems. However, these systems do not always work as expected and might exert excess forces before stopping [1]. Furthermore, in some situations, like a robot located in a surgery theatre, collisions are not acceptable, and full collision avoidance should be implemented. This coincides with the goals of the Industry 4.0 concept [2].

A crucial part for the obstacle avoidance is getting real-time measurements of the workspace and environment around the robot. Such sensing can be done using a variety of sensors: laser scanners, mono and stereo vision, RGB-D cameras, ultrasound sensors and motion capture systems.



(a) UR3, UR5, UR10 (b) KUKA LBR iiwa (c) Franka Emika Panda

Fig. 1. Robot manipulators used in our experiments.

Even with advanced sensing systems, the problem still stands in the requirement of having a reliable calibration between the sensors and the robot - so-called Eye-to-Hand calibration [3]. Such a calibration maps the coordinate frames of the robot and vision sensors into a common coordinate frame. As a result, the position of an obstacle detected by one of the sensors can be easily calculated in point of view of the robot, and the necessary action is taken to avoid it. There are reliable and even automatic ways of performing Eye-to-Hand calibration, however, if any of the sensors is unexpectedly moved in relation to the robot, and unaccounted for, the calibration becomes invalid and the system might malfunction [4]. This can be an issue in dynamic environments like a surgery theatre, where there is a lot of human movement, as well as the equipment is constantly shifted around. Similar works and research on dynamic obstacle avoidance for robot arms normally require a fully-calibrated robot-camera system, which can be a challenge in non-static configuration setups [5] [6].

We have shown that the transfer learning approach can be used to adapt the system trained to recognise and estimate the position of the robot base and joints from one robot model to a new unseen one by having a limited amount of data [7] [8]. We base our work on a previously trained multi-objective CNN on Universal Robots (UR) and extend our work in the following manner. Instead of adapting the network to the new robot type, we adjust the CNN to incorporate new robot types, while still being able to recognise previously trained robots. Eventually, the proposed system is capable of identifying 5 different robots. Furthermore, with the help of motion capture system tracking the camera, we collected a complex training datasets with the camera being moved around in an unconstrained manner, obtaining a variety of viewing angles of the robots in front of complex backgrounds. A more thorough analysis also shows the impact of the accuracy

[1 4 5 6]Justinas Mišeikis, Kyrre Glette, Ole Jakob Elle and Jim Torresen are with the Department of Informatics, University of Oslo, Oslo, Norway
[2 3] Inka Brijačak and Saeed Yahyanejad are with the Joanneum Research - Robotics, Klagenfurt am Wörthersee, Austria
[4 6]Kyrre Glette and Jim Torresen also have affiliation with RITMO, University of Oslo
[5]Ole Jakob Elle has his main affiliation with The Intervention Centre, Oslo University Hospital, Oslo, Norway oelle@ous-hf.no
[1 4 6] {justinm,kyrrehg,jimtoer}@ifi.uio.no
[2] Inka.Brijacak@joanneum.at
[3] Saeed.Yahyanejad@joanneum.at

depending on the distance between the camera and the robot.

This paper is organized as follows. First, we provide an overview of related work in Section II. We present the system setup and dataset collection in Section III. Then, we explain the proposed method and CNN structure and configuration in Section IV and the transfer learning procedure in Section V. We provide experiments and results in Section VI, followed by relevant conclusions and future work in Section VII.

II. RELATED WORK

With the recent deep learning revolution in computer vision, especially for classification tasks, like ImageNet, it has been proven that it is possible to learn to identify objects in difficult environments and conditions [9].

In order to train a deep learning network, large amounts of training data are needed with precisely marked ground truth data. Collecting such training datasets can be a time-consuming task. However, transfer learning approach is useful when a fully trained system exists for one type of the problem and can be adapted for different datasets by adjusting some of the parameters of the network while keeping other parameters fixed [9]. This has been proven to work for mid-level image representations in object classification, using the pre-trained network on natural images to adapt for medical image recognition and even emotion recognition [10] [11] [12]. Another interesting application of transfer learning is to use a fully trained network on night-time satellite imagery of poverty areas and adapt it to recognise poverty areas from daytime satellite imagery [13]. Furthermore, detailed analyses of the transfer learning approaches were made with surveys of the techniques used and various CNN structures [14] [15].

Moreover, CNN based work in the field of human pose estimation in 2D [16], known as *OpenPose*, allowed further improvements on 3D human pose estimation with the help of a depth sensor [17]. The accuracy for a human keypoint in 3D is around $11cm$, mainly due to the inaccuracy of the depth sensor which grows with distance from the sensor.

On the other hand, many purely geometrical techniques have been employed to determine the position and orientation of an object from a single image by using some prior knowledge about the target object [18] [19]. In general, with these methods, they try to find patterns and features such as edges and corners which match the expected model and accordingly estimate the position and orientation. Some other researchers exploited the existence of a 3D model such as a CAD model [20] [21] to increase the accuracy of the estimation. Although the precision of their method is higher compared to our CNN-based method, they mainly suffer from a major drawback: they can only perform with solid and rigid objects which clearly does not apply to robot manipulators. Another problem is the necessity of having a 3D model available beforehand, which in our method is substituted with the training procedure. However, our method performs more robustly in case of deviation from the model in case of physical damages or attached end-effectors, and it can also use the image colour information which is normally missing in a 3D model.

III. SYSTEM SETUP AND DATASET COLLECTION

Deep learning networks are capable of robustly recognising objects in complex backgrounds, but in order to achieve good performance, a large amount of precisely marked and diverse training data is needed. Considering the setup of three heterogeneous robotic manipulators, a system had to be set up to generate training data with accurate ground truth data marked automatically, given that manual ground truth generation for such datasets would take up a significant amount of time and effort.

Our setup consists of the following three robot types:

- Universal Robots: UR3, UR5, UR10, 6 DoF, Figure 1(a)
- KUKA LBR iiwa - 7 R800, 7 DoF, Figure 1(b)
- Franka Emika - Panda, 7 DoF, Figure 1(c)

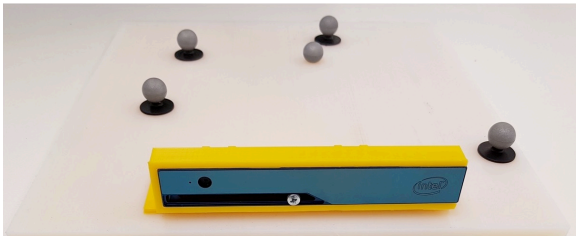


Fig. 2. Setup with the Optitrack Motion Capture System

This two-stage transfer learning work, as the basis, uses already trained multi-objective CNN [7], which was trained on datasets containing all three robot models from UR. Datasets containing KUKA LBR iiwa were previously collected for our one-stage transfer learning project [8]. All these datasets were collected using Kinect V2 sensor with necessary Eye-to-Hand calibration [22] every time position of the camera changed relative to the robot-base in order to achieve a high variety of backgrounds.

TABLE I. Dataset summary describing a number of samples collected for each type of the robot.

Robot Type	Number of Datasets	Total Number of Samples
Universal Robots	9	4350
Kuka LBR iiwa	14	1837
Franka Panda	5	2513

Table I summarizes all datasets collected for each robot with their number of recordings where they differ by camera placement relative to the robot, illumination, background.

New datasets containing Franka Emika Panda robot were recorded with free-moving Intel RealSense R200 RGB-D camera instead of Kinect V2. Since Eye-to-Hand calibration is only valid for the fixed camera setups, we could not use this method for the camera to robot coordinates-transformation measurements. Instead of performing Eye-to-Hand calibration, we have placed *Optitrack* (Motion Capture System) [23] markers over the moving camera and around the base of the robot in order to bring both systems into one coordinate frame of Optitrack (Figure 2). Since Optitrack's marker (*Rigid-Body* or *Rig*) origin is not exactly aligned with

Fig. 3. Samples from the collected robot datasets. Robots used are Universal Robots (silver-blue), Kuka LBR iiwa (silver-orange) and Franka Emika - Panda (white-black). A variety of robot configurations, camera movements and angles as well as lighting conditions were used. In some cases, even other robots in the background were present.

the camera's optical frame origin, extrinsic calibration was performed as described in [24] by observing and detecting one additional rig, that was fixed in the Optitrack frame, with our RGB camera from multiple positions. Example frames taken from the whole dataset can be seen in Figure 3.

Once all the transformations are connected in one coordinate system, a precise robot mask that is separating a robot body from the background when overlaying a colour image, used as a ground truth for teaching the CNN, can be calculated. It is done automatically by using ROS with *MoveIt* package [25]. The robot model, taken from the Robot Description Format (URDF) files and mesh files provided by the robot manufacturers, is updated with the live information from robot's joints encoder readings to create robot shape in real-time [26]. This shape is transformed to depth camera's coordinate frame and mask image is constructed.

In order to ensure the robustness of the system, robot movements were programmed so that each robot joint is moved through the full range of motion in combination with other joints taking into account self-collision and table collision avoidance. Also, a trigger signal is used to save the data after each robot movement. With each trigger, we save camera colour images, robot joints and Cartesian coordinates, as well as ground-truth robot mask images. Moreover, to ensure a perfect overlap between colour and depth images, internal extrinsic camera calibration was used. All the input images are also rectified and have the resolution of 480×360 pixels. Testing and validation sets were divided by the ratios of 80% and 20% respectively based on random sampling.

IV. CNN STRUCTURE AND CONFIGURATION

The structure of a multi-objective CNN is identical to previous work, where it was trained on UR robots [7]. The network trains for multiple outputs simultaneously by taking a single 2D colour image as an input. The network in this paper is trained on four objectives: Robot mask, Robot type, 3D Robot base position and 3D Position of the robot joints.

The network has multiple branches, with some of the convolutional layers shared and then branched off to optimise specifically for each of the objectives. The structure of the multi-objective CNN can be seen in Figure 4.

A. Loss Functions

The loss function is needed to define the quality of training and drive the CNN towards achieving better results. Given four different outputs of the network, four loss functions need to be defined and eventually combined in a single one for the training process. Firstly, we describe each one of them and then explain how they are connected together.

Normally, the robot body takes up a rather small area in the whole image. For UR datasets, the area of the robot body is between $6-17\%$, for Kuka datasets, it is between $8-18\%$ and for Franka Panda, it is between $5-22\%$. Given a standard approach for the pixel classification loss function, an accuracy of over 78% could be reached by classifying the whole image as background. This is conceptually wrong, so the loss function is adapted by calculating the foreground weight w_{fg} as defined in Equation 1. It is based on the inverse probability of the foreground and background classes, where $Z \in \{fg, bg\}$.

$$w_{fg} = \frac{1}{\mathsf{P}(Z = fg)} \tag{1}$$

The background weight w_{bg} is calculated in Equation 2.

$$w_{bg} = \frac{1}{\mathsf{P}(Z = bg)} \tag{2}$$

Definition of the loss function for the robot mask is done in two steps. First, a per-pixel loss l^n is calculated in

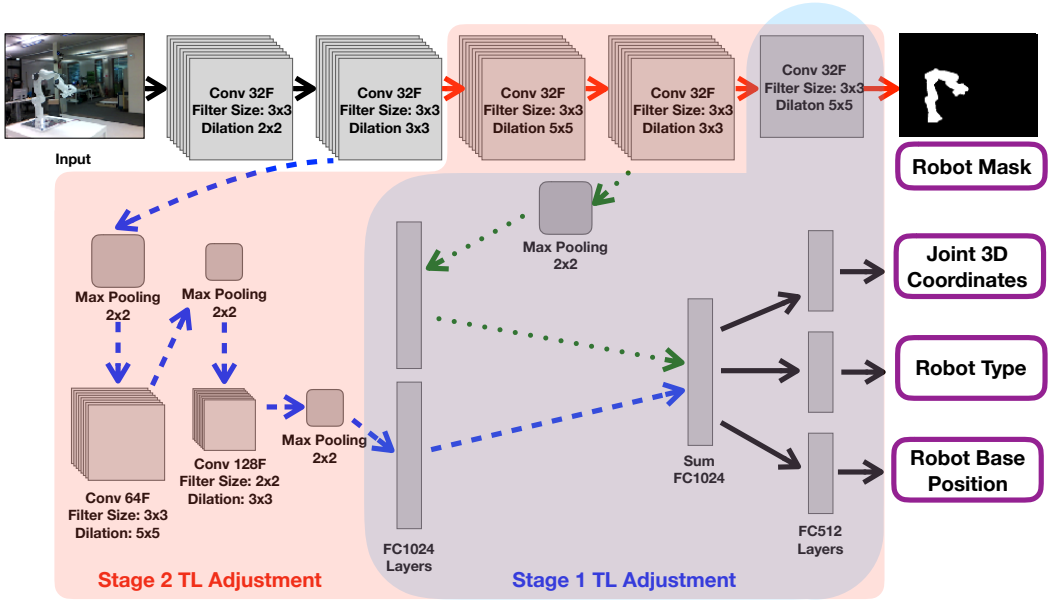


Fig. 4. The multi-objective CNN with a two-stage transfer learning. The CNN is optimising simultaneously for four objectives at the same time: robot mask, 3D coordinates of robot joints, 3D coordinates of the robot base and the robot type. The network is taught in two stages using the transfer learning approach. In stage 1, the parameters for all the layers, besides the final ones marked in blue are *frozen* and the system is trained until there is no more improvement. Afterwards, in stage 2, the parameters marked in red, as well as all the stage 1 layers, are adjusted during the training. This approach allows faster training compared to the full training, while still reaching good accuracy.

Equation 3, where i_{est} is $\mathrm{P}(Y = fg)$, $(1-i_{est})$ is $\mathrm{P}(Y = bg)$ and i_{gt} is the ground truth value from the mask image.

$$\begin{aligned} l^n(I^n_{est}, I^n_{gt}) = &- w_{fg} i_{gt} \log(i_{est}) \\ &- w_{bg}(1 - i_{gt}) \log(1 - i_{est}) \end{aligned} \tag{3}$$

Then, a normalised loss calculation is done for the whole image $\mathcal{L}_{mask}$ in Equation 4. In order to keep the same learning parameters independent of the input image size, a normalisation factor $\mathcal{N}$ is used, which is a number of pixels in the image.

$$\mathcal{L}_{mask}(I_{est}, I_{gt}) = \frac{1}{\mathcal{N}} \sum_n l^n(i_{est}, i_{gt}) \tag{4}$$

Estimation of the 3D coordinates of the robot base and robot joints are defined as a regression problem instead of classification. Loss function uses the Euclidean distance between the ground truth and estimated values by the CNN. For the robot joints estimation, the loss function $\mathcal{L}_{Jcoords}$ is described in Equation 5, where N_j is the number of joints, J_i is the ground truth position of each joint and E_i is the estimated values by the neural network.

$$\mathcal{L}_{Jcoords} = \frac{1}{N_j} \sum_{i=1}^{N_j} \|J_i - E_i\|_2 \tag{5}$$

The loss function for the coordinates of the robot base $\mathcal{L}_{Bcoords}$ is calculated in Equation 6. B_{xyz} is the ground truth position of the robot base in 3D, and E_{xyz} is the estimated 3D position of the robot base. These positions are relative to the coordinate frame of the camera.

$$\mathcal{L}_{Bcoords} = \left\|B_{xyz} - E_{xyz}\right\|_2 \tag{6}$$

A multi-class categorical cross-entropy approach is used to identify the robot type $\mathcal{L}_{type}$. $\mathcal{L}_{type}$ is calculated in Equation 7, where p is the ground truth labels, q are the predicted labels and $c \in R$, where R contains all the available types of robots in the dataset.

$$\mathcal{L}_{type} = -\sum_c p(c) \log q(c) \tag{7}$$

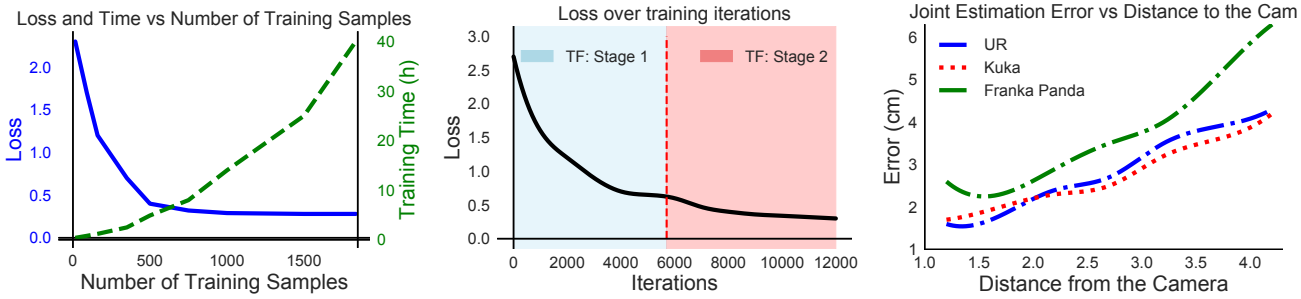
Eventually, all four previously defined loss functions are combined into a single loss function to be used in the training of the multi-objective CNN. The final loss function $\mathcal{L}_{final}$ is calculated as a weighted sum of all the loss functions, as shown in Equation 8. The larger the weight W, the higher the impact on the corresponding value.

$$\begin{aligned} \mathcal{L}_{final} = &W_{mask}\mathcal{L}_{mask} + W_{Jcoords}\mathcal{L}_{Jcoords} \\ &+ W_{Bcoords}\mathcal{L}_{Bcoords} + W_{type}\mathcal{L}_{type} \end{aligned} \tag{8}$$

V. TRAINING USING TRANSFER LEARNING

A common approach to training such a system would be to train the whole system using full datasets of all the robots. However, this would take a significant amount of computation and time. Overall, the goal of this work is to analyse the possibility of having a pre-trained system and expand it to include more robot types while having a limited amount of training data and time.

Transfer learning allows us to use a fully trained system for one robot type and then adjust it to include the newly provided training data. This is done by *freezing* the parameters in some of the layers while adjusting the remaining ones. Given this partial adjustment, the training time and amount of training data required can be significantly reduced.



(a) Loss function and training time against the number of training samples used. This was acquired by running a number of experiments using input datasets of different size. Using more than 500 training samples per robot type does not give a significant accuracy benefit while increasing the training time.

(b) Loss function against the number of training iterations. Stage 1 and stage 2 of our two-stage transfer learning approach are marked by background colours on the graph. It can be seen that when stage 1 training saturates, unlocking parameters of more CNN layers allow the network to further improve results in stage 2 training.

(c) Errors for 3D position estimation of robot joints depending on the camera distance from the robot. Results are grouped by the robot type. Error and distance have close to linear correlation, but Franka Panda has higher error compared to the other robots. This is due to the robot body, which gives low contrast compared to the background.

Fig. 5. Evaluation of the two-stage transfer learning method using the test dataset in various categories.

The system was fully trained using UR robots with Kuka and Franka Panda robots added using the transfer learning approach. One crucial difference is that UR robots have 6 joints, while Kuka and Franka Panda are 7 joint robots. In general, it has been found that first convolutional layers tend to learn general visual features, while further layers figure out specific visual cues of the objects. Both, UR and Kuka robots have bright coloured joint covers, while the rest of the robot is silver, however, Franka Panda is mainly black and white, as seen in Figure 1.

Due to these differences, a two-stage transfer learning approach was taken up, as shown in Figure 4. In the first step, just the final layers of the multi-objective CNN are trained. This allows the network to adjust the dense layers to select the best-learned features for the robot recognition using currently learned visual cues. Only a small part of the CNN is adjusted, so the learning process is fast, and it re-learns robot classification and position estimation using current convolutional layer parameters.

However, after some point the learning process saturates and no more improvements are observed, defined by the reduction of loss. At this stage, the second part of the CNN is unlocked, allowing to modify parameters for the additional convolutional layers. This results in modifications of the visual cues that are learned as well as adjusting the final dense layers. The training speed is slower compared to the first stage of learning, but the loss is reduced even further.

In order to add the new robot types using the transfer learning approach, the new training dataset has to include both, the robot that the network was originally trained on, as well as the new robot(s) that should be recognised.

Weights for the loss function are adjusted to give more impact to identifying mask and robot type compared to our previous work. Selected weight values, based on trial and error from a number of experiments, were as follows: W_{mask}: 1.2, $W_{Jcoords}$: 1.2, $W_{Bcoords}$: 1.2 and W_{type}: 0.6.

The number of training samples varied by the experiment and the input size of the images was scaled down and cropped to 256×212 pixels for all the datasets. The pixel intensity values of the input images were normalised to the range between 0 and 1. The learning rate was set to 0.001 at the start of the training and then gradually decreased towards 0.000001 as the training progressed.

VI. EXPERIMENTS AND RESULTS

The main goal of the experiments was to evaluate the capability of including new robot types by using the two-stage transfer learning method while using a multi-objective CNN fully trained on UR robots as a starting point.

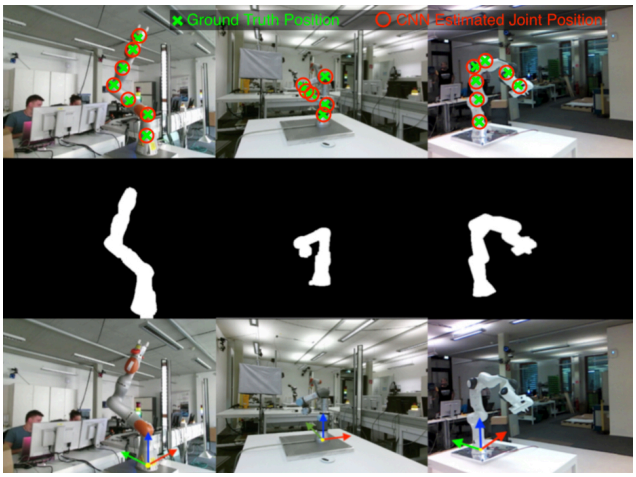


Fig. 6. Visualisation of the output from the presented multi-objective CNN trained using a two-stage transfer learning approach. Each column represents each robot type in the following order: Kuka, UR and Franka Panda. The first row shows the estimated 3D joint position (red circles) against the ground truth position (green crosses). The second row shows the estimated mask of the robot and the third row shows the estimated robot base position.

Each of the experiments was conducted by taking a different size transfer learning dataset using a randomised sample selection to maximise the diversity of the data. The maximum amount of data was limited by the Kuka robot dataset to ensure the same amount of samples in each test for each of the robot types.

The system was evaluated using a testing set by comparing the output against the ground truth data. The robot mask accuracy is defined by counting the number of pixels in the

CNN output image that match the ground truth mask. For the robot joint and base coordinates, the Euclidean distance between the CNN estimated results and ground truth results was calculated. We compare the results between each of the robot type in a number of categories. Results are summarised in Table II and visualisation of estimations plotted on top of the testing set samples can be seen in Figure 6.

TABLE II. Summary of the Transfer Learning results (using 500 samples of each robot type for training) on the test set.

Measure	UR	Kuka	Franka Panda
Mask Accuracy, %	97.0%	$\mathbf{97.7\%}$	94.3%
Robot Type Accuracy, %	97.5%	$\mathbf{100\%}$	96.1%
Joint Pos Error (Median)	$\mathbf{3.12cm}$	$3.30cm$	$3.64cm$
Base Pos Error (Median)	$2.42cm$	$\mathbf{2.36cm}$	$3.01cm$

All of the robots had joint 3D position estimation error under $3.64cm$, with the base position estimation error under $3cm$. The mask accuracy estimation exceeded 97% for UR and Kuka robots, while for Franka Panda it was a bit lower at 94.3%. Robot type was recognised correctly in all of the cases for Kuka robot, while UR and Franka Panda recognition were 97.5% and 96.1% respectively. Overall, it was noticed that given the distinct features, the CNN performed the best on Kuka robot, while low contrast Franka Panda robot had the worst results, but not far behind.

Considering overall performance of the two-stage transfer learning, as shown in Figure 5(a), for the multi-objective CNN, it can be seen that the loss function stops improving when having datasets of size between 500 and 750 training samples for each of the robot type, which corresponds to 7 to 10 hours of training time. Increasing the number of training samples beyond 750 does not improve the learning process, but significantly increases the training time.

Compared to the previously presented work in [7], the detection accuracy of the current two-stage transfer learning approach achieved similar accuracy in a joint position error of $2.46cm$ vs $3.12cm$ and slightly worse accuracy for the robot mask estimation: 97% vs 98% in the previous work. The full training time of the multi-objective CNN for UR robots took 60 hours vs 10 hours in the current work.

The performance of each training stage of transfer learning is shown in Figure 5(b). Stage 1, with parameters in final CNN layers being adjusted, saturates after 6000 iterations. Afterwards, further layers are *unlocked* switching to Stage 2 and the loss function reduces even further settling down between $10000 - 12000$ iterations.

Furthermore, we analyse the impact of the joint and base position estimation depending on the distance between the camera and the robot, visualised in Figure 5(c). There is close to a linear relationship between the distance between the robot and the accuracy of the 3D position estimation of the robot joints. Interestingly, at a very close distance of 1.2 meters, Franka Panda robot shows worse performance compared to 1.5 meter distance.

The detection time or forward-propagation of the multi-objective CNN was measured to be $19 - 23ms$ per frame on a nVidia GTX 1080 Ti graphics card.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a two-stage transfer learning approach, which allows to re-train a previously trained multi-objective CNN to include numerous new robot types using a limited amount of training data. This approach reduces the time spent on collecting datasets with ground truth data, as well as the training time of the network itself. Furthermore, a concept of a multi-objective CNN capable of identifying heterogeneous robots, classifying their types and estimating 3D positions of their joints and base was proven. A simple 2D colour image was used as an input and Kuka and Franka Panda robots were mounted with two-finger grippers on the end-effector, which were not taken into account by the CNN. The network successfully estimated the position of the robot as the was no end-effector present. If the TCP of the end-effector would be required, it could be calculated by adding the necessary CAD model or offset information to the estimated position of the end-effector.

With the detection time of under $23ms$, the system has proven to be capable of working in real-time. At the current stage, a powerful GPU is needed to run it, however, a goal of optimising it for smaller mobile systems could be pursued. In this case, it could be implemented in small wearable cameras to be used both, for mobile robots or for human operators working in a robotised environments and used as a safety system, which can detect possible collisions without having any direct communication between the devices. The outcome could be a valuable measure for various safety applications in *Human-Robot Interaction* scenarios, where we need to know the position of the human and robot and their individual joints respective to each other.

The achieved robot joint position estimation is not accurate enough for visual servoing operations, but future work can focus on accuracy improvements. We believe that by using higher resolution images, multi-sensor detection and tracking in time series, the accuracy of our system could be improved. Furthermore, an analysis of the impact on the detection accuracy depending on the weight selection of loss functions and layer selection for the transfer learning will be done. With the current results, a high-level control is still possible for human-robot and robot-robot interaction.

Additionally, with the given robot mask detection in a 2D image, some robot self-inspection could be done to detect any damage, especially for autonomous robots operating in remote or disaster areas, where people do not have access to, for example for planetary exploration rovers.

ACKNOWLEDGMENT

This work is partially supported by The Research Council of Norway as a part of the Engineering Predictability with Embodied Cognition (EPEC) project, under grant agreement 240862, and Research Council of Norway through its Centres of Excellence scheme, project number 262762, and by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) within the project framework CollRob (Collaborative Robotics).

REFERENCES

[1] I. Bonev, "Should We Fence the Arms of Universal Robots?" http://coro.etsmtl.ca/blog/?p=299, ETS, 2014, accessed September 5, 2018.

[2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[3] R. K. Lenz and R. Y. Tsai, "Calibrating a cartesian robot with eye-on-hand configuration independent of eye-to-hand relationship," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 9, pp. 916–928, 1989.

[4] J. Miseikis, K. Glette, O. J. Elle, and J. Torresen, "Automatic calibration of a robot manipulator and multi 3d camera system," in *System Integration (SII), 2016 IEEE/SICE International Symposium on*. IEEE, 2016, pp. 735–741.

[5] J. Mainprice and D. Berenson, "Human-robot collaborative manipulation planning using early prediction of human motion," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 299–306.

[6] J. Mišeikis, K. Glette, O. J. Elle, and J. Torresen, "Multi 3D camera mapping for predictive and reflexive robot manipulator trajectory estimation," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–8.

[7] J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, "Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images," in *2018 15th International Conference on Ubiquitous Robots (UR)*, June 2018, pp. 597–603.

[8] J. Mišeikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, "Transfer learning for unseen robot detection and joint estimation on a multi-objective convolutional neural network," in *2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, Aug 2018, pp. 337–342.

[9] 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.

[10] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[11] H. Greenspan, B. van Ginneken, and R. M. Summers, "Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, 2016.

[12] H.-W. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler, "Deep learning for emotion recognition on small datasets using transfer learning," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*. ACM, 2015, pp. 443–449.

[13] S. M. Xie, N. Jean, M. Burke, D. B. Lobell, and S. Ermon, "Transfer learning from deep features for remote sensing and poverty mapping," *CoRR*, vol. abs/1510.00098, 2015. [Online]. Available: http://arxiv.org/abs/1510.00098

[14] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[15] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, 2016.

[16] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *CVPR*, 2017.

[17] C. D. W. B. Christian Zimmermann, Tim Welschehold and T. Brox, "3d human pose estimation in rgbd images for robotic task learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. [Online]. Available: https://lmb.informatik.uni-freiburg.de/projects/rgbd-pose3d/

[18] C.-M. Cheng, H.-W. Chen, T.-Y. Lee, S.-H. Lai, and Y.-H. Tsai, "Robust 3d object pose estimation from a single 2d image," in *Visual Communications and Image Processing (VCIP), 2011 IEEE*. IEEE, 2011, pp. 1–4.

[19] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis, "Single image 3d object detection and pose estimation for grasping," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3936–3943.

[20] C.-Y. Tsai, W.-Y. Wang, C.-H. Huang, and B.-R. Shih, "Cad model-based 3d object pose estimation using an edge-based nonlinear model fitting algorithm," in *Proceedings of the 3rd IIAE International Conference on Intelligent Systems and Image Processing 2015*. IIAE, 2015, pp. 59–62.

[21] J. J. Lim, A. Khosla, and A. Torralba, "Fpm: Fine pose parts-based model with 3d cad models," in *European conference on computer vision*. Springer, 2014, pp. 478–493.

[22] R. Y. Tsai and R. K. Lenz, "A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration," *Robotics and Automation, IEEE Transactions on*, vol. 5, no. 3, pp. 345–358, 1989.

[23] N. Point, "Optitrack," *Natural Point, Inc.,[Online]. Available: http://www. naturalpoint. com/optitrack/.[Accessed 5 5 2018]*, 2011.

[24] S. Chiodini, M. Pertile, R. Giubilato, F. Salvioli, M. Barrera, P. Franceschetti, and S. Debei, "Camera rig extrinsic calibration using a motion capture system," in *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. IEEE, 2018, pp. 590–595.

[25] I. A. Sucan and S. Chitta, "MoveIt!" *Online Available: http://moveit.ros.org*, 2013.

[26] W. Meeussen, J. Hsu, and R. Diankov, "URDF-Unified Robot Description Format," 2012.

Bibliography

[1] J. Wallén, *The history of the industrial robot*. Linköping University Electronic Press, 2008.

[2] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[3] K. D. Mankoff and T. A. Russo, "The kinect: a low-cost, high-resolution, short-range 3d camera," *Earth Surface Processes and Landforms*, vol. 38, no. 9, pp. 926–936, 2013.

[4] J. E. Colgate, J. Edward, M. A. Peshkin, and W. Wannasuphoprasit, "Cobots: Robots for collaboration with human operators," 1996.

[5] "Infographic: A brief history of collaborative robots > engineering.com," https://www.engineering.com/AdvancedManufacturing/ArticleID/12169, (Accessed on 11/13/2018).

[6] J. Roach and M. Boaz, "Coordinating the motions of robot arms in a common workspace," *IEEE Journal on Robotics and Automation*, vol. 3, no. 5, pp. 437–444, 1987.

[7] J. Leitner, S. Harding, M. Frank, A. Förster, and J. Schmidhuber, "Transferring spatial perception between robots operating in a shared workspace," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1507–1512.

[8] E. A. Merchán-Cruz and A. S. Morris, "Fuzzy-GA-based trajectory planner for robot manipulators sharing a common workspace," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 613–624, 2006.

[9] J. Y. Lew, Y.-T. Jou, and H. Pasic, "Interactive control of human/robot sharing same workspace," in *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2000, pp. 535–540.

[10] C. Breazeal, C. D. Kidd, A. L. Thomaz, G. Hoffman, and M. Berlin, "Effects of nonverbal communication on efficiency and robustness in human-robot teamwork," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 708–713.

[11] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 338–345.

[12] C. Lenz, M. Grimm, T. Röder, and A. Knoll, "Fusing multiple kinects to survey shared human-robot-workspaces," *Technische Universität München, Munich, Germany, Tech. Rep. TUM-I1214*, 2012.

[13] R. Hayne, R. Luo, and D. Berenson, "Considering avoidance and consistency in motion planning for human-robot manipulation in a shared workspace," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3948–3954.

[14] J. M. Zurada, *Introduction to artificial neural systems*. West publishing company St. Paul, 1992, vol. 8.

[15] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[16] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[18] 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.

[19] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[22] S. J. Pan, Q. Yang, *et al.*, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[23] S. Hoo-Chang, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, p. 1285, 2016.

[24] N. Point, "Inc.: Optitrack-optical motion tracking solutions," 2009.

[25] 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 *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[26] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[27] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[28] I. A. Sucan and S. Chitta, "Moveit!" *Online at http://moveit. ros. org*, 2013.

[29] S. Edwards and C. Lewis, "Ros-industrial: applying the robot operating system (ros) to industrial applications," in *IEEE Int. Conference on Robotics and Automation, ECHORD Workshop*, 2012.

[30] T. T. Andersen, "Optimizing the universal robots ros driver," *Technical University of Denmark, Department of Electrical Engineering*, 2015.

[31] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[32] OpenCV, "Camera Calibration and 3D Reconstruction," *Online Available: http://docs.opencv.org/modules/calib3d/doc/ camera_calibration_and_3d_reconstruction.html*, 2015.

[33] P. Fankhauser, M. Bloesch, D. Rodriguez, R. Kaestner, M. Hutter, and R. Siegwart, "Kinect v2 for mobile robot navigation: Evaluation and modeling," in *2015 International Conference on Advanced Robotics (ICAR)*. IEEE, 2015, pp. 388–394.

[34] S. Ma and Z. Hu, "Hand-Eye Calibration," in *Computer Vision*. Springer, 2014, pp. 355–358.

[35] R. Horaud and F. Dornaika, "Hand-Eye Calibration," *The international journal of robotics research*, vol. 14, no. 3, pp. 195–210, 1995.

[36] J. Bohren, R. B. Rusu, E. G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mösenlechner, W. Meeussen, and S. Holzer, "Towards autonomous robotic butlers: Lessons learned with the pr2," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 5568–5575.

[37] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010.

[38] 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.

[39] "A comprehensive guide to convolutional neural networks — the eli5 way," https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, (Accessed on 01/08/2019).

[40] H.-W. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler, "Deep learning for emotion recognition on small datasets using transfer learning," in *Proceedings of the 2015 ACM on international conference on multimodal interaction*. ACM, 2015, pp. 443–449.

[41] "PHOENIX CONTACT | Homepage Corporate Website," https://www.phoenixcontact.com/, (Accessed on 14/11/2018).

[42] “Robots as ’pump attendants’: Tu graz develops robot-controlled rapid charging system for e-vehicles,” https://www.tugraz.at/en/tu-graz/services/news-stories/tu-graz-news/singleview/article/robots-as-pump-attendants-tu-graz-develops-robot-controlled-rapid-charging-system-for-e-vehicles/, (Accessed on 01/09/2019).

[43] R. Meng, J. Perez-Ramirez, and H. Moustafa, “Localizing a vehicle’s charging or fueling port-methods and apparatuses,” Feb. 14 2019, uS Patent App. 16/020,740.

[44] J. Miseikis, P. Knobelreiter, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, “Robot localisation and 3d position estimation using a free-moving camera and cascaded convolutional neural networks,” in *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, July 2018, pp. 181–187.

[45] J. Miseikis, I. Brijacak, S. Yahyanejad, K. Glette, O. J. Elle, and J. Torresen, “Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images,” in *2018 15th International Conference on Ubiquitous Robots (UR)*, June 2018, pp. 597–603.