FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Self-adaptive Cobots in Cyber-Physical Production Systems

Roberto de Nóbrega Nogueira

MSC DISSERTATION

Supervisor: Gil Manuel Magalhães de Andrade Gonçalves

Co-Supervisor: João Pedro Correia dos Reis

Co-Supervisor: Rui Pedro Ferreira Pinto

February 19, 2019

Abstract

Absolute automation in certain industries, such as the automotive industry, has proven to be disadvantageous. Robots are fairly capable when performing tasks that are repetitious and demand precision. However, during the manufacturing processes, some tasks are either too expensive or exceptionally difficult to automate. These types of tasks require the flexibility of a human operator. The current paradigm for manufacturing places the human operator in a "slave" status in relation to the robots but with the 4th Industrial Revolution, an opportunity to change this model presents itself. This new paradigm focuses on creating a hybrid solution comprised of the adaptability and resourcefulness of humans and the precision and efficiency of machines. This type of hybrid solution is able to support safe physical cooperation between operators and manipulators. To achieve this kind of cooperation, new manipulators have been designed with characteristics that are favourable for collaboration such as force limited joints and limited working velocity. Despite robotic manufacturers' best efforts to create collaborative solutions, these lack self-adaptability and true collaborative behaviour. And so, vision systems have been integrated with these manipulators to enhance their potential dynamic behaviour. These systems can provide a better perception of the environment the manipulator inhabits and they also support complex interactions such as understanding commands through gestures, reinforcing the human as the control element in the process.

During this dissertation, a vision-based collaborative proof-of-concept framework was created using the Kinect v2, MATLAB and a UR5 manipulator.

In this framework, three behavioural modes are implemented for the manipulator, a Self-Adaptive mode that implements collision avoidance through the use of repulsive action and Region Growing segmentation, a Collaborative mode that allows physical interaction between the user and the robot, by tracking and approaching the hand of the user, and finally, a Safe mode that places the manipulator in a predefined position. These modes are activated through gestures with recourse to the Kinect's body tracking and gesture recognition algorithm.

For extrinsic parameter estimation, the Kabsch algorithm is shown to be very competent with the calibration method employed. Region Growing segmentation in the depth space, for manipulator self-recognition, is shown to be more flexible in relation to 3D model-based segmentation. The benefits of repulsive action for collision avoidance are discussed for unmapped environments. Gesture control is also shown to be an effective communication tool for human-robot interaction.

At the end of this dissertation, the proof-of-concept is validated for a future real-time implementation with a few recommendations.

Acknowledgements

I would like to thank all who have contributed to the accomplishment of this dissertation:

- To my supervisors Professor Gil Gonçalves, João Reis, and Rui Pinto for all their time and knowledge;
- To the Systec FOF team for their support and patience;
- To my team in LSTS for all the advice;
- To António Sérgio Ferreira for the motivation and clarity to push through;
- To André Diegues for his patience and willingness to help me;
- To my friends who motivated and helped me throughout my academic life;
- To Beatriz Santos, Vítor Cabral, and my mother Lília de Nóbrega Nogueira for their unconditional love and support.

Thank you.

Roberto de Nóbrega Nogueira

*“So here we are reinventing the wheel
I’m shaking hands with a hurricane
It’s a colour that I can’t describe,
It’s a language I can’t understand
Ambition, tearing out the heart of you
Carving lines into you
Dripping down the sides of you”*

Verse from Pioneers by Bloc Party

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	2
1.4	Problem Definition	2
2	State of the art	5
2.1	Smart Manufacturing	5
2.1.1	Cyber-Physical Production Systems	6
2.1.2	Self-Adaptive Systems	8
2.2	Collaborative Robotics	8
2.2.1	Human-Robot Interaction	10
2.2.2	Human-Machine Interfaces	11
2.3	Visual Servoing	12
2.3.1	Kinect v2	13
2.4	Calibration	16
2.4.1	Pinhole Camera Model	17
2.4.2	Perspective Projection	20
2.4.3	Kabsch Algorithm	21
2.5	Forward and Inverse Kinematics	22
2.5.1	Denavit-Hartenberg (DH) Convention	25
2.6	Image Processing	27
2.6.1	Image Filtering	27
2.6.2	Segmentation	29
2.7	Vision-Based Interaction and Obstacle Detection	30
3	Framework	33
3.1	System Architecture	33
3.2	Communications	34
3.2.1	Decoding Data Stream	36
3.2.2	Formatting Commands	37
3.3	System Calibration	38
3.3.1	Forward Kinematics	38
3.3.2	Intrinsic Parameters	40
3.3.3	Extrinsic Parameters	42
3.3.4	Validating the parameters	45
3.4	Implemented Modes	48
3.4.1	Self-Adaptive Mode	48

3.4.2	Collaborative Mode	53
3.4.3	Safe Mode	54
4	Results and Scenarios	55
4.1	Code Execution Time	55
4.2	Gesture Recognition	56
4.2.1	Gesture Detection Time	56
4.2.2	False Recognition	57
4.3	Scenarios	57
4.3.1	First Scenario	57
4.3.2	Second Scenario	61
4.4	User Learning Curve	63
5	Discussion of Results	65
5.1	System Limitations	65
5.2	Performance	65
5.3	Gesture Detection	66
5.4	Region Growing for Segmentation	66
5.5	Repulsive Vector for Collision Avoidance	67
5.6	User Learning Curve Analysis	67
6	Conclusion and Future Work	69
6.1	Conclusions	69
6.2	Future Work	71
	References	73

List of Figures

2.1	5c Architecture	6
2.2	Architecture of the SelSus CPPS [1]	7
2.3	Industry 4.0 Ecosystem [2]	8
2.4	Kuka Industrial Manipulator	9
2.5	ABB Dual Arm Collaborative Robot	10
2.6	Levels of Autonomy based on HRI	10
2.7	Kinect v2	12
2.8	Zed Stereo Camera	12
2.9	Asus Xtion Pro	13
2.10	Kinect v2 Component Location	13
2.11	Joint Identification [3]	14
2.12	Closed Hand State	15
2.13	Open Hand State	15
2.14	Lasso Hand States (both hand positions are recognized as lasso)	15
2.15	Kinect v2 Coordinate System [4]	16
2.16	Pinhole Camera Model [5]	17
2.17	Extrinsic and Intrinsic Parameters	18
2.18	Image Plane Coordinate Frame	18
2.19	Skew [6]	19
2.20	Radial Distortion [6]	19
2.21	Moustache Radial Distortion [7]	20
2.22	Tangential Lens Distortion [6]	20
2.23	Perspective Projection Geometry [8]	21
2.24	Kinematic Diagram with the assigned axis	23
2.25	Forward vs. Inverse Kinematics	24
2.26	DH Parameter Table	26
2.27	Spatial Filtering	27
2.28	Gaussian Kernel	27
2.29	Gaussian Filter Applied	28
2.30	Non-Linear Filter	28
2.31	Median Filter for Salt and Pepper Noise	28
2.32	Semantic Segmentation from ICNet	29
2.33	Common Neighbourhood Connectivity Masks	29
2.34	Tesla's Autopilot Vision System	30
3.1	System Architecture	33
3.2	Communication Architecture	34
3.3	Data stream	36

3.4	Practical example in MATLAB	37
3.5	UR5 Kinematic Diagram following the DH Convention [9]	38
3.6	DH parameters for UR5	39
3.7	Camera Positions	40
3.8	Kinect Calibration Setup	41
3.9	GML Toolbox Interface	41
3.10	Relation between the Kinect frame and the World frame	42
3.11	End-effector Coordinates in UR5 interface	43
3.12	Depth Image of UR5	43
3.13	Finding end-effector centre in IR image	44
3.14	Extracting the depth value for end-effector centre	44
3.15	Output from the Validation Test	45
3.16	Joint positions calculated by FK	46
3.17	End-effector pixel position and depth value extraction	46
3.18	State Machine Diagram for Implemented Modes	48
3.19	Self-Adaptive Structure Diagram	49
3.20	Depth Frame to Segmented Frame	50
3.21	Surveillance Area	51
3.22	Obstacle in Safe Volume Marked	52
3.23	Repulsive Vector Illustration	53
3.24	Collaborative Mode Example	54
4.1	Average Run-time Bar Graph	55
4.2	Correct/Incorrect Gesture Recognition	57
4.3	Example 1	58
4.4	Example 2	58
4.5	Example 3	58
4.6	Example 4	59
4.7	Example 5	59
4.8	Example 1 for Safe Mode	59
4.9	Example 2 for Safe Mode	60
4.10	Example 3 for Safe Mode	60
4.11	Example 4 for Safe Mode	60
4.12	Example 5 for Safe Mode	60

Abbreviations

2D	2 Dimensional
3D	3 Dimensional
3R	3-Revolute
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
AOA	Agente Oriented Architecture
CPS	Cyber-Physical System
CPPS	Cyber-Physical Production System
CPU	Central Processing Unit
Cobot	Collaborative Robot
DH	Denavit-Hartenberg
DoF	Degrees of Freedom
FK	Forward Kinematics
FPGA	Field-Programmable Gate Array
fps	frames per second
GML	Graphics and Media Lab
GPU	Graphics Processing Unit
HMI	Human-Machine Interface
HRI	Human-Robot Interaction
IoT	Internet of Things
IR	Infrared
LoA	Level of Autonomy
MRI	Magnetic Resonance Imaging
RGB	Red Green Blue
RGB-D	Red Green Blue - Depth
SBC	Single Board Computer
SDK	Software Development Kit
SOA	Service-Oriented Architecture
ToF	Time of Flight
UTF	Unicode Transformation Format
VR	Virtual Reality

Chapter 1

Introduction

1.1 Context

With the rise of the 4th Industrial Revolution, a new set of ideas came into play, one of them being that digital, physical and biological systems should be intertwined through a cloud-based network called the Internet of Things (IoT). This connection allows interoperability, information transparency and the decentralization of decision making, in order to create a whole dynamic system. Additionally, excessive automation has proven to no longer be the future for most industries, but instead a hybrid solution where humans and robots coexist, communicate and cooperate to an extent that allows a fluid and safe work environment.

IoT and robotics are no longer restrained to the industry, IoT devices flood households and simple robots start to help with tasks such as cleaning or gardening. As industrial and military technologies evolve, domestic technology accompanies this progression. This means that more and more non-robotic experts will come to interact with these machines at home and in the workplace, which leads to a necessity of collaborative robotic solutions that, are capable of interacting safely with humans who do not have specialized training to be able to deal with these devices.

Although there has been a combined effort to create a fluid interaction between humans and robots, machines lack a very important characteristic that allows humans to act dynamically upon their surroundings: predicting events based on their senses and adapting their actions depending on that prediction. Besides this adaptive, reactive nature, human interaction is very complex and requires a plethora of sensory, cognitive, emotional and social skills, some of which robots still do not possess. Bestowing robots with the necessary senses and skills to have a collaborative and self-adaptive nature will considerably improve safety and production efficiency in industrial environments, not only as a consequence of the robots' abilities but also the perspective of the worker to view the robot as its peer, and not just a machine. This type of technology presents an opportunity to develop and integrate complex robotic solutions in other sectors such as healthcare, domestic and defence.

1.2 Motivation

Humans are, without a doubt, very complex and dynamic beings. They can learn, train and adapt to unforeseen situations. This grants humans the ability to invent and improvise. These abilities are great assets for industries, however, humans are not precise enough and require rest. Robots, on the other hand, are precise systems that can be programmed to repeat tasks continuously but they are not able to adapt or improvise when confronted with a situation that is not contemplated in their programming.

The current paradigm for manufacturing does not take full advantage of the capabilities of humans and robots. It tries to maximize robotic tasks and assigns humans with tasks that are too expensive or difficult to automate. This creates a bottleneck in the process if the human operator is not able to keep up with the speed of the line. Also, if some type of manufacturing error occurs, the robots are not able to correct it and the human operator is expected to do so.

Industry 4.0 presents an opportunity to shift this paradigm to one that can exploit the full potential of both humans and robots, by placing the human as the control element for the system, in certain tasks. This means that robots could interact with the operators according to their needs and ultimately collaborate in the accomplishment of tasks. This new paradigm will place human ingenuity in control of robotic precision, effectively making use of the strengths of both humans and machines.

1.3 Objectives

The main objective of this dissertation is to create a self-adaptive, vision-based collaborative framework to be integrated with a collaborative robotic arm, where the human is the control element. This will be achieved by creating a "language" between the user and the manipulator through gestures. To allow this type of nomenclature between the two entities, the body tracking and gesture recognition capabilities of the Microsoft Kinect v2 will be employed. To support physical interaction and reactive behaviour from the manipulator, the depth map created by the Kinect v2 will be related to the physical space captured by the sensor through geometry. Reactive behaviour will be implemented through the use of artificial repulsive vectors upon detection of obstacles. Since the manipulator will exist in the space captured by the Kinect, image processing will be exploited to extract important information from the depth map, specifically, the Region Growing segmentation algorithm will be tested for the recognition of the manipulator. By the end of this dissertation, the concept will be evaluated in relation to real-world implementation in Cyber-Physical Production Systems.

1.4 Problem Definition

According to the objective of this dissertation, creating vision-based interaction between a manipulator and a human can be decomposed into smaller specific sequential problems. The first

problem is related to communication since the data that the Kinect and the manipulator provide need to be captured. For the manipulator, communication is done through the TCP/IP protocol. This allows the main computer of the system to send and receive data from the manipulator. The Kinect v2 communicates through USB 3.0. In order to utilize the Kinect, the Kinect for Windows Software Development Kit must be installed.

Now that the different types of data are collected in one device, the data needs to be related. What exists in image space, is a projection of the physical space captured. Meaning they are related through geometry. To solve this problem firstly a world frame of reference must be chosen, then the image data must be related to the Kinect's space and from the Kinect space related to the chosen world frame. Afterwards, the manipulator data must also be related to the world frame. Once all the data is transformed into the same frame of reference it can be compared.

With the data related, the self-adaptive and collaborative behaviours can be implemented. The self-adaptive behaviour for the manipulator will be created using artificial repulsive forces once an obstacle is detected through vision, however, the problem of the manipulator being in the scene captured by the vision sensor emerges. Since the manipulator does not know what it is in the image, it will consider itself as an obstacle. To try to solve this problem, the Region Growing algorithm for segmentation will be utilized in order to remove the manipulator from the image, and thus the robot will "recognize" itself. The collaborative behaviour will utilize the body tracking information from the Kinect, so the manipulator can perceive where the human is in space to allow safe physical interaction.

Once the behaviours are complete, the Kinect supported gestures will be mapped to each behaviour and this way the control of the system is human-centred, achieving a robot that has visual perception of the workspace, and can act and cooperate according to the user's needs.

Chapter 2

State of the art

2.1 Smart Manufacturing

In recent years, technologies such as IoT, Big Data, CPS, 3D Printing and Collaborative Robotics have matured into a stage where the Industry is now interested in implementing them in factories. Powerful Single Board Computers (SBC) have allowed embedded systems to reach higher levels of complexity, being Internet connectivity (either wired, wireless or both) an essential specification.

3D Printing has made fast and easy prototyping at lower costs achievable. If any error is detected during this phase, a new design can be made and printed without changing any physical aspect of the process, meaning no tool or mould reshaping is needed. Furthermore, this technique is no longer solely used for prototyping but also to manufacture parts, that are deemed viable, in consumer-grade products [10].

Cyber-Physical System architectures have allowed the virtualization of all entities involved in the process into the digital realm [11]. This virtualization creates a digital identity, a digital-twin, also called Smart Component, that replicates its physical counterpart's characteristics. When connected to a network, this type of system allows seamless communication about its (in reality the physical entities) status, decentralization of decision making and higher levels of dynamic automation. For a general CPS, a 5-level architecture was proposed in [12], named 5c architecture, represented in Figure 2.1. 5c architecture was created as a guideline for the creation of CPS, but there is still no standard for CPS architecture, so there are several other proposals such as in [13], where there is a division of the CPS into four layers:

1. Perception Layer: Where the physical devices are located.
2. Network Layer: Where the communication protocols are implemented.
3. Middleware Layer: Where the data processing occurs.
4. Application Layer: Where the data that has been obtained can be monitored.

Figure 2.1: 5c Architecture

Big Data methods are needed to be able to manage and process the massive amounts of real-time data created by CPS. As an example, anomaly detection in a CPS using Big Data methods has been proposed in [14].

The intertwining of these technologies, with new management ideologies, is referred to as Smart Manufacturing, and it is part of the ongoing 4th Industrial Revolution that is working to achieve efficient, adaptable and safer industries.

2.1.1 Cyber-Physical Production Systems

A Cyber-Physical Production System (CPPS) is a specific application of CPS in an industrial environment. These types of systems are fundamental to enable Human-Machine Collaboration. As the CPPS follows the principles of CPS in the virtualization of physical entities and networking, it creates opportunities for flexible assembly lines. As an example, a Cobot could be receiving and interpreting biometric data from its fellow human coworker, as proposed in [15], in order to understand levels of fatigue or stress and adapt to the present conditions. Another is a learning factory that focuses on interconnecting product development and the actual manufacturing of the product, such as it has been proposed in [16].

As in CPS, the architecture of CPPS is fundamental for its applicability in real-world manufacturing. Communication protocols and interfaces, internal and external processing capabilities, shop-floor equipment, data payload, Human-Machine Interfaces (HMI) are some of the many factors to be held accountable in the design of CPPS architectures. As introduced in [1], a hybrid CPPS capable of dealing with collaborative environments is established, exploring Service-Oriented Architectures (SOA) and Agent-Oriented Architectures (AOA), granting efficient handling of dynamic shop-floors. In Figure 2.2, the proposed architecture is represented.

Figure 2.2: Architecture of the SelSus CPPS [1]

In [17] a more complex definition of CPPS can be found: "A Cyber-Physical Production System is a composition of human resources, production equipment and aggregated products towards which it establishes one or several cyber-physically formulated interaction interfaces. These interfaces are used for monitoring and control of the CPPS operations as well as to tap into the knowledge generated both by the human resources, and the equipment, during the production process as well as knowledge generated by its aggregated products throughout their life-cycle. This internal knowledge is used in different time scales to continuously improve operations and to inform the strategic consumptions of capital, raw materials and energy in their many forms. Cyber-physically formulated products will also generate value for external systems, as part of networks of things and services, towards which they maintain interaction interfaces. The outcome of such interactions is external knowledge. Access to it may be offered as a value-adding service that can potentially help to further improve CPPS's operations."

CPPS are fundamental to create an industrial ecosystem on which every entity, that is an element of the process, is aware and can communicate with all the other constituting parts as shown in Figure 2.3.

The Industry 4.0 ecosystem

Figure 2.3: Industry 4.0 Ecosystem [2]

2.1.2 Self-Adaptive Systems

Self-Adaptability is a specification that has become a must in systems that belong to the Industry 4.0, but over millions of years of evolutionary history, nature has created and perfected the most dynamic, self-adaptive mechanisms, that ensure the efficient survival of biological beings in an ever-changing environment. In light of Biomimicry, innovation in self-adaptive systems must try to explore nature's time-tested patterns and strategies. Neural Networks are a good illustration of how emulating natural structures can revolutionize and improve current technology.

Self-Adaptability can be understood as the ability of a system to autonomously adjust its behaviour according to external stimuli, and furthermore, as a part of being self-adaptive, the system must be able to understand if the adjustment was, or still is, beneficial according to current conditions.

2.2 Collaborative Robotics

In today's industry, there is a very clear distinction between industrial manipulators and force limited manipulators [18]. Although either can be collaborative, in the sense that, according to very strict safety measures, they may work and cooperate with humans, force limited manipulators are specially designed to be able to interact with their human co-workers, in contrast to industrial manipulators that require safety cages and light curtains to create a physical barrier that prevents interaction. The former type of robot can normally be referred to as Cobots. Besides being force limited, these manipulators possess built-in force sensors that allow impact sensing so that, in case of a collision, the manipulator can acknowledge it and cease its activity.

The design of Cobots tends to be round so that the force of an impact is spread over a larger area and it takes attention to pinch points in order to avoid any compression related injuries during interactions. Often padded or cushioned designs are used to minimize potential damage to both humans and the manipulators. Differences in construction and design are visually noticeable in Figure 2.4 and Figure 2.5.

Even though the design and sensorization of these Cobots allow some level of collaboration, there are still several limitations associated with off-the-shelf solutions. The major limitation is related to how the robots respond to a collision. Current Cobots, such as the UR5, do not have the ability to actively avoid collisions without external hardware and software. Meaning that the robot will stop its movement when in contact with the object, instead of recalculating a path that avoids the collision.

Figure 2.4: Kuka Industrial Manipulator

Figure 2.5: ABB Dual Arm Collaborative Robot

2.2.1 Human-Robot Interaction

Human-Robot Interaction (HRI) is the field that seeks to understand and shape the interactions between a single or various humans and one or more robots. These interactions are heavily influenced by the level of autonomy the robot possesses, the nature of the information exchange, the ability of the human and the robot to learn, train and adapt to the task at hand.

The Level of Autonomy (LoA) of the robotic device is not the main focus of HRI but it is crucial to understand how it will influence collaborative tasks. A robotic manipulator, to be truly collaborative and self-adaptive, must own a dynamic autonomous nature, meaning it will shift its LoA to adapt to any given situation. This is what is referred to as Peer-to-Peer collaboration, which may be harder to implement than full autonomy because to achieve this kind of dynamic behaviour, the robot must understand and possess some amount of situational awareness. An in-depth survey by Michael A. Goodrich and Alan C. Shultz [19] further describes interaction attributes and architecture approaches in HRI.

Figure 2.6: Levels of Autonomy based on HRI

HRI is also shaped by the way humans perceive the robot. Identifying what role the robot has in relation to the person it is interacting with is important to understand how this same person

will interact with it [20, 21]. For example, a robot that is perceived as a peer will have different interactions with its co-workers than a robot that is a mentor in relation to the human.

The idea that someday complex AI-driven robots might coexist in houses and city streets to help humans with tedious tasks, children's education or elderly care, is closer to becoming a reality. This brings to light several ethical issues, some that have been made popular by means of literature or cinema [22], and others that have appeared with experience or accidents. These preconceived ideas of what "*intelligent*" robots can, or will be, is ultimately affecting the way laws and policies are designed, which has direct consequences in HRI. While not so ethically driven, some defend that mutual understanding will be crucial for the smooth integration of robots in our society [23], where robots understand patterns of human behaviour, such as micro-expressions, gaze, gestures, language or even biometrics, so that they can infer on emotional state and intent. Adding to this, not only should robots have the ability to understand humans, but also be able to be understood and to be, in some sense predictable so whomever they might interact with can easily understand their aim.

2.2.2 Human-Machine Interfaces

Human-Machine Interfaces is a category that includes any type of device or software, that allows a Human to interact with a machine. This can range from a push-button and a screen to a fully fledged voice and gesture recognition system. With IoT becoming more and more prevalent in current society, the way humans interact with electronic devices is changing and because of this, so are the interfaces that control them.

One of the biggest promises in HMI, that is still in research, is the ability to control prosthetics with a Brain-Machine Interface (BMI). This type of technology has the ability to revolutionize society, possibly solving tremendous setbacks in current medicine associated with paraplegia and artificial limbs. More on this topic can be read in [24].

Industry 4.0 has involved technologies such as Augmented Reality (AR) in manufacturing environments. In AR graphical, sound and tactile feedbacks are superimposed over reality, in order to enhance the worker's efficiency. In [25] an illustrative example of a potential application of AR to an assembly line can be viewed.

AR is not to be confused with Virtual Reality (VR). In VR applications, the user is completely immersed in a computer-generated simulation, whereas in AR the user is aware of their surroundings and virtual information is layered upon the real-world. Each of these technologies has its rightful place in the industry. While AR applications, such as proposed in [26, 27], have been focused on assisting and guiding workers through a task, VR applications are generally aimed at improving employee training, as proposed in [28, 29].

There have been great efforts to create HMI systems with gesture and speech recognition. The development of this type of technology has allowed the creation of smart-devices and software like Amazon Alexa, Xbox Kinect, Siri and Google Translator. This is also an important step towards collaborative robotics as speech and gesture control can be applied, such as proposed in [30], or,

going even further, with a system based on continuous gesture recognition for anticipating human intent, as proposed in [31].

2.3 Visual Servoing

Robot visual servoing is a technique that integrates visual data into the control system of the robot. Systems which possess visual sensors are more flexible in dynamic environments, since this type of sensor can capture a wider variety of events. A comprehensive survey comparing visual servoing methods can be found in [32], where further information on the categories and application areas of these methods is thoroughly examined.

Visual servoing methods are greatly advantageous for collaborative applications since humans are physically expressive. Even though bio-metric sensors can feed information about blood pressure or heart rate, they cannot perceive posture, facial expressions or gestures.

RGB-Depth (RGB-D) cameras and Stereo cameras are top contenders for these applications since they not only return an image, but also depth information. These types of technologies allow 3D reconstruction of the scene which can provide an even wider overview of the environment the robot is placed. Some examples of these types of cameras are:

- The Kinect v2 is an RGB-D camera. The depth image is captured by emitting infrared (IR) light and measuring the time it takes to return to the infrared sensor. This technology is called Time of Flight (ToF).

Figure 2.7: Kinect v2

- The Stereo Labs Zed is a stereo camera. This camera does not use infrared light for depth measurement, but actually captures two RGB images at the same time, from two different and physically separated RGB cameras. The distance between the cameras is known and so, a depth map can be constructed by comparing the differences in the images.

Figure 2.8: Zed Stereo Camera

- The Asus Xtion Pro is also an RGB-D camera but, unlike the Kinect v2, the technology utilized to capture depth information is based on structured light. The principle of this

technology is to project an infrared pattern of light onto the scene and evaluate, by means of an infrared sensor, how the pattern has been deformed, thus returning the depth information of the scene.

Figure 2.9: Asus Xtion Pro

2.3.1 Kinect v2

As mentioned above the Kinect v2 is an RGB-D camera. It was originally developed to allow human motion and voice-controlled interfaces and games for the Xbox One. The Kinect v2 has an RGB camera, an IR emitter and camera, which together create a depth sensor. The IR camera can also be used to obtain infrared images.

Figure 2.10: Kinect v2 Component Location

Recommended Hardware Configuration [33]:

- 64-bit (x64) processor
- 4 GB Memory (or more)
- Physical dual-core 3.1 GHz (2 logical cores per physical) or faster processor
- USB 3.0 controller dedicated to the Kinect for Windows v2 sensor
- DX11 capable graphics adapter
- A Microsoft Kinect v2 sensor, which includes a power hub and USB cabling

Infrared/Depth camera	Resolution	512 x 424
	Field of View	70.6° x 60°
	Angular Resolution	0.14° per pixel
	Operating Range	0.5 to 8 meters
	Maximum Reliable Range for Body tracking	4.5 meters
RGB Camera	Resolution	1920 x 1080
	Field of View	84.1° x 53.8°
Frame Rate		30 Hz
Connection type		USB 3.0
Dimensions	width x depth x height (mm)	249 x 66 x 67

Table 2.1: Kinect v2 Hardware Specification

Despite its specifications, the hardware is not what makes the Kinect v2 special. The accompanying Windows Software Development Kit (SDK) 2.0 is what makes this device so powerful. The SDK provides gesture and voice recognition, up to 6 person body tracking, facial expression tracking, lean tracking and 3D Point Cloud reconstruction [34]. This SDK runs only on Windows Operating Systems (OS), specifically Windows 8 and above. For this reason, the Kinect was hacked and open source drivers were created that support Linux, Windows and Mac OS [35], however, these drivers only support depth data, RGB data, IR data and the registration between both types of data for 3D Point Cloud reconstruction.

2.3.1.1 Body Tracking and Gesture Recognition

The Kinect's body tracking algorithm is based on the work of Shotton et al. and is described in [36]. It can track up to 6 bodies each with 25 joints. In Figure 2.11 the tracked joints plus their identification name and number can be seen:

Figure 2.11: Joint Identification [3]

The Kinect also supports gesture recognition. It can identify the state of each hand in the following positions:

Figure 2.12: Closed Hand State

Figure 2.13: Open Hand State

Figure 2.14: Lasso Hand States (both hand positions are recognized as lasso)

There are two more states implemented, but they do not translate to a physical gesture and are purely informative, which are Unknown and NotTracked.

2.3.1.2 Camera, Depth and Colour Space

The result of any tracking algorithm that operates in 3D, such as body tracking, is stored in reference to the camera space coordinate system. The Kinect's coordinate system, as shown in Figure 2.15, has its origin at the centre of the IR camera and is defined as the Z-axis growing out in the direction the camera is facing, the X-axis grows to the left of the IR camera, in the direction

of the emitter, and the Y -axis grows upward, according to the tilt of the Kinect. The unit of this reference frame is in meters.

Figure 2.15: Kinect v2 Coordinate System [4]

Depth Space is the 2.5-dimensional space, or pseudo-3D, that can be imagined as a matrix of rows and columns, where each value within the rows and columns represent another dimension. In a depth image there are 512 columns and 424 rows, which represent the pixels of the image, and each pixel value is the distance from the camera, measured in millimetres, to the object being perceived in that pixel. This space has a 2-dimensional frame that expresses the location of the origin for the pixels. The origin is on the top-left corner of the image, as it is the standard and will later be seen in Figure 2.18. The values of the measurement range from 0 to 8000 mm. Since the IR image and the depth image are created using the same sensor, the infrared image has the same frame characteristics, except the pixel value does not represent a distance, but instead an IR value. Meaning a pixel $(row_i, column_i)$ in the depth image is the same $(row_i, column_i)$ in the IR image.

The RGB camera has a slight offset from the IR sensor, meaning that the images produced by these sensors are two individual views of the same scene. Besides this spatial offset, the colour space has 1920 columns and 1080 rows and three channels, Red, Green and Blue, meaning it has considerably more pixels. The origin of the image frame follows the standard, being located in the top-left corner of the image.

To be able to create a 3D reconstruction of a scene, for example, the RGB pixel values need to be correlated with the corresponding Depth pixel values. Since they are expressed in relation to different frames a process called image registration has to be utilized. Image Registration is the process that transforms distinct data sets into a single coordinate system. The Kinect already provides a registration module called *Coordinate Mapper* that can relate either the depth data in relation to the colour data, or vice-versa, and much more [37].

2.4 Calibration

In robotics, specifically robotic manipulators, there is a necessity to describe position and orientation of the end-effector in relation to a well-defined frame, the world frame. The world frame will identify the origin of the whole space considered. Normally, this space is represented by the Cartesian Coordinate System, which is one way to describe Euclidean Space. In this representation system, a point $P = (x, y, z)$ is defined by 3 coordinates that express the Euclidean distance of

P to 3 orthogonal planes defined by the axes of the reference frame. The base of the manipulator is often chosen to be the reference frame and it will establish the origin for the whole kinematic chain. Furthermore, in robotic applications, it is common to add 3 more coordinates to the distance coordinates (x, y, z) that express the orientation of the end-effector in relation to the same 3 planes.

Robotic manipulators and cameras, as every physical object, inhabit 3-dimensional space. However digital images do not exist in the same 3-dimensional space. They are 2-dimensional projections of a 3-dimensional scene, meaning there is a relation between them but they are not directly comparable. Since it is required to work with both images and physical objects, there needs to be coherence when relating the two entities.

2.4.1 Pinhole Camera Model

According to [5] the model of a pinhole camera consists of an *image plane* I , a *focal plane* F , which are parallel to each other, and a point called the *optical centre* C that belongs to F .

Figure 2.16: Pinhole Camera Model [5]

The *focal length* is defined as the distance between C and I . The *optical axis* f , is the line that is perpendicular to the *focal* and *image plane* and passes through the *optical centre*. The intersection of f with the *image plane* is called the *principal point* c .

To be able to use this model, two sets of parameters are needed:

Figure 2.17: Extrinsic and Intrinsic Parameters

- The intrinsic parameters, which relate the position of a point in the coordinate frame of the *focal plane* and the position of its projection in the *image plane*. These parameters can be expressed in the form of a matrix $\mathbf{K}$:

$$\mathbf{K} = \begin{bmatrix} fx & s & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

Where fx and fy are the *focal lengths*. In a true pinhole camera $fy = fx$, which should be the same as f that was mentioned above, but in practice, these values can differ for a variety of reasons, such as flaws in the digital sensor and unintentional distortion from the lens.

cx and cy are the *principle point offset* which depends on the origin of the *image plane*. Usually, the origin is defined at the upper left corner, with the y -axis going down and the x -axis going to the right. This means that cx and cy are the displacement values of the *principle point* according to the frame of the *image plane* shown in Figure 2.18.

Figure 2.18: Image Plane Coordinate Frame

The pixel skew s is defined as:

$$s = fy \cdot \tan \alpha \quad (2.1)$$

Where α is the angle between the vertical image axis and the axis that is perpendicular to the horizontal image axis, as shown below:

Figure 2.19: Skew [6]

The pinhole model is useful but it does not take into account lens distortion because an ideal pinhole camera does not possess a lens. Modern cameras use lenses and so, two more intrinsic parameters exist [6]:

– Radial Distortion:

Radial distortion takes place when light bends more near the edges of a lens than its centre. This type of distortion can be barrel or pincushion as shown in Figure 2.20:

Figure 2.20: Radial Distortion [6]

There is still one more type of radial distortion which is a combination of both the barrel and pincushion distortion, called wavy or moustache distortion, shown in Figure 2.21.

Figure 2.21: Moustache Radial Distortion [7]

- Tangential Lens Distortion:

Tangential Distortion is only present if the lens and the *image plane* are not parallel as demonstrated in Figure 2.22.

Figure 2.22: Tangential Lens Distortion [6]

- The extrinsic parameters, which relate the position of a point in the coordinate frame of the *focal plane* to the position of the same point in the world frame of reference. These parameters can be expressed in the form of a matrix E :

$$E = \begin{bmatrix} R & t \end{bmatrix}$$

Where R is a 3 by 3 rotation matrix and t is a 3 by 1 translation vector in relation to the world frame.

2.4.2 Perspective Projection

According to [8], a simple way to derive the equations that relate 3D projections to 2D space is by using perspective projection geometry.

Figure 2.23: Perspective Projection Geometry [8]

A point in 3D space, $P_{cartesian} = (x_{cartesian}, y_{cartesian}, z_{cartesian})$, has its 2D projection $Pixel = (x', y')$, given that we have the *focal length* value f , equal to:

$$x' = \frac{x_{cartesian} \cdot f}{z_{cartesian}} \quad y' = \frac{y_{cartesian} \cdot f}{z_{cartesian}} \quad (2.2)$$

But recalling Section 2.4.1, it is known that each axis can, in fact, have distinct *focal lengths*, resulting in:

$$x' = \frac{x_{cartesian} \cdot fx}{z_{cartesian}} \quad y' = \frac{y_{cartesian} \cdot fy}{z_{cartesian}} \quad (2.3)$$

Also, the *principle point offset* has to be considered since the frame origin of the *image plane* is not the *principle point*:

$$x' = \frac{x_{cartesian} \cdot fx}{z_{cartesian}} + cx \quad y' = \frac{y_{cartesian} \cdot fy}{z_{cartesian}} + cy \quad (2.4)$$

Equations 2.4 will provide the approximately correct pixel point, (x', y') given a point in camera space.

2.4.3 Kabsch Algorithm

Presented in [38], with a later correction in [39], the Kabsch algorithm was created to help find the best "rotation to fit a given atomic arrangement to approximately measured coordinates.". This means that this algorithm calculates the rotation matrix between two paired sets of points so that the root mean squared deviation is minimized.

The mathematical description of this algorithm is quite complex. However, it has been implemented in several programming languages such as MATLAB [40], Python [41] and C++ [42]. A quick overview of how this algorithm works, according to the MATLAB implementation, is described below:

- Given two paired sets of points, P and Q:

$$P = \begin{bmatrix} x_0 & y_0 & z_0 \\ \dots & \dots & \dots \\ x_n & y_n & z_n \end{bmatrix}, \quad Q = \begin{bmatrix} h_0 & u_0 & v_0 \\ \dots & \dots & \dots \\ h_n & u_n & v_n \end{bmatrix}$$

- The sets of points are translated to the same frame of reference by subtracting their respective centroid:

$$P = P - p_{centroid} \quad Q = Q - q_{centroid} \quad (2.5)$$

- The Covariance matrix is calculated:

$$C = PQ^T \quad (2.6)$$

- The Covariance matrix is decomposed through Singular Value Decomposition:

$$C = U \cdot S \cdot V \quad (2.7)$$

- Next if $|VU^T| > 0$, the rotation matrix is updated:

$$R = V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} U^T \quad (2.8)$$

And the translation vector is given by:

$$\vec{t} = q_{centroid} - R \cdot p_{centroid} \quad (2.9)$$

2.5 Forward and Inverse Kinematics

Robotic manipulators abound in shape, size, task specificity, payload and degrees of freedom (DoF). The knowledge needed to design such a mechanism ranges from Mechanical Engineering, Electrical Engineering, Control Theory to Physics. The design of a manipulator, such as actuator and sensor placement, will directly influence how difficult it is to control [43]. Kinematics is the selection of geometric principles and techniques that focus on position, velocity and acceleration of particles. In robotics, this translates to understanding the end-effector's position and/or orientation in relation to a reference frame, given certain values to its joint variables. This is also known as Forward Kinematics (FK), also sometimes called Direct Kinematics. To obtain the Forward Kinematics of a manipulator a kinematic diagram, as shown in Figure 2.24, must be drawn. Following, the axis of the kinematic chain must be assigned. There are methods that help assign the axis, the most common being the Denavit-Hartenberg Convention that will be explained in Section 2.5.1.

Figure 2.24: Kinematic Diagram with the assigned axis

The kinematic diagram will aid in the search for the rotation matrix and translation vector between the designated frames of each joint. Once these are achieved, a Homogeneous Transformation matrix can be constructed. Given the rotation matrix and the translation vector:

$$R = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad d = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \quad (2.10)$$

The Homogeneous Transformation matrix will be:

$$H = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d_x \\ d & e & f & d_y \\ g & h & i & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.11)$$

The Homogeneous Transformation matrix is a useful tool since it reduces the mathematical burden behind rigid motions to simple matrix multiplication [44], as shown in Equation 2.18 in Section 2.5.1. Notice that to use Homogeneous Transformation matrices, points must be augmented by adding a fourth element, as such:

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2.12)$$

Now given a point $P^1 = [x, y, z]^T$ represented in reference to frame 1, it should then be extended:

$$P^1_{extended} = [x, y, z, 1]^T$$

Assuming the Homogeneous Transformation matrix between frame 1 and frame 0 is known, the same point can be expressed in relation to frame 0 as:

$$P^0_{extended} = H^0_1 \cdot P^1_{extended} \quad (2.13)$$

In contrast, Inverse Kinematics is the study of a manipulator's joint variables in relation to a known end-effector position and/or orientation. Which means that the end-effector of the manipulator has a target point, with a certain angle, and through Inverse Kinematics the collection of joint values is obtained.

Figure 2.25: Forward vs. Inverse Kinematics

More often than not, depending on the DoF of the manipulator, there are several combinations of joint variables values that lead to the same end-effector target point, this redundancy in robotic manipulators must be taken into account and studied thoroughly.

There is another problem to consider with kinematics called singularities. A singularity is a point that forces the manipulator's joint velocities to tend to infinity and is related to the Jacobian Matrix of a robot. The Jacobian Matrix allows the transformation from the manipulator's joint velocities to end-effector velocities:

$$\xi(t) = J\dot{q}(t) \quad (2.14)$$

A singularity occurs when the determinant of the Jacobian of the manipulator equals to zero, meaning the Jacobian becomes rank deficient:

$$\det(J) = 0 \quad (2.15)$$

The system of equations that relate the joint velocities to the end-effector velocity contains linear dependencies, this results in the existence of either no solutions for the system, or an infinite number of them [45]. A physical visualization of typical singularities in a 6 DoF Cobot can be observed in [46]. There are ingenious ways of avoiding singularities, even though robotic

manufacturers have created countermeasures to avoid the manipulator from breaking, or getting stuck, which normally relies on stopping the manipulator or reducing actuation speeds, for shop-floor applications this is sometimes disadvantageous. So, in order to reduce the probability of encountering a singularity, many have learned to add a slight angle to the tooling process, noting that this does not eliminate singularities but it can, in some cases, avoid them.

Generally, Inverse Kinematics is considered more demanding than Forward Kinematics. This is because of the extensive geometric and trigonometric equations that have to be calculated, added to the already complex matrix algebra involved. Given this complexity, many have sought to create fast computational methods, to help mitigate this issue, such as proposed in [47]. Others describe the existing mathematical methods for general Inverse Kinematic calculation [48].

For the Universal Robotics UR5 manipulator there are several papers that describe the Inverse Kinematics [9, 49, 50].

2.5.1 Denavit-Hartenberg (DH) Convention

To achieve the position and orientation of the end-effector accurately it is imperative to have the correct rotation matrices and translation vectors that express each joint of the manipulator regarding a world frame. The higher the number of DoF the manipulator possesses the more complex the mathematical procedure to obtain the Homogeneous Transformation matrix becomes, so a convention was presented in [51], that simplifies greatly the mathematics of the procedure. By following a set of rules when assigning frame coordinates to the manipulator, it is possible to get the Denavit-Hartenberg parameters that will then be fed into a generalized Homogeneous Transformation matrix. The set of rules for coordinate frame assignment is comprised of:

- Rule 1: The Z-axis must be the axis of rotation for a revolute joint or the direction of motion for a prismatic joint;
- Rule 2: The X-axis must be perpendicular both to its own Z-axis and the Z-axis from the frame preceding it;
- Rule 3: All frame must follow the right-hand rule;
- Rule 4: Each X-axis must intersect the Z-axis of the frame that precedes it.

After the implementation of these rules, the DH parameters can be found. It is usual to construct a table as shown in Figure 2.26:

Link i	$\theta_i(\text{rad})$	$\alpha_i(\text{rad})$	$a_i(\text{m})$	$d_i(\text{m})$
1	θ_1	α_1	a_1	d_1
2	θ_2	α_2	a_2	d_2
3	θ_3	α_3	a_3	d_3
...	...	...	...	...
...	...	...	...	...
N	θ_n	α_n	a_n	d_n

Figure 2.26: DH Parameter Table

Where:

- θ_i is defined as the rotation around Z_{i-1} that is required so that X_{i-1} matches X_i ;
- α_i is defined as the rotation around X_i that is required to get Z_{i-1} to match Z_i ;
- a_i is defined as the distance from the origin of frame _{i} to the origin of frame _{$i-1$} along the X_i axis ¹;
- d_i is defined as the distance from the origin of frame _{$i-1$} to the origin of frame _{i} along the Z_{i-1} direction.

Once the parameters have been obtained, they can be plugged into the Homogeneous Transformation matrix formulated by the convention. The DH convention states that each Homogeneous Transformation matrix H_n , where n is the number of the joint, is represented as the product of four basic transformations, detailed further in [44]:

$$H_n = \text{Rotation}_{z,\theta_n} \cdot \text{Translation}_{z,d_n} \cdot \text{Translation}_{x,r_n} \cdot \text{Rotation}_{x,\alpha_n} \quad (2.16)$$

Resulting in the base Homogeneous Transformation matrix for the DH parameters:

$$H_n = \begin{bmatrix} \cos \theta_n & -\sin \theta_n \cos \alpha_n & \sin \theta_n \sin \alpha_n & r_n \cos \theta_n \\ \sin \theta_n & \cos \theta_n \cos \alpha_n & -\cos \theta_n \sin \alpha_n & r_n \sin \theta_n \\ 0 & \sin \alpha_n & \cos \alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.17)$$

It is important to notice that the Homogeneous Transformation matrix H_n , is in relation to the previous frame, meaning that in order to calculate the coordinates in relation to frame 0 of a point expressed in frame 2, for example, the Homogeneous Transformation matrix properties can be used to get the matrix that correlates the two spaces:

$$H^0_2 = H^0_1 \cdot H^1_2 \quad (2.18)$$

¹ Sometimes a can be referred to as r to avoid confusion with α (alpha)

2.6 Image Processing

For any vision-based system, the way the data is prepared and processed is very important. Methods and operations that retrieve relevant information from images are the major interest in the image processing field. Image processing has been massively developed in medical imaging, to aid in the automatic detection of abnormalities that might be diagnosed in imaging-based exams, such as Magnetic Resonance Imaging (MRI).

2.6.1 Image Filtering

Image filtering is usually associated with the pre-processing phase of image processing. Filtering can help with noise reduction or removal, edge detection and feature enhancement. Spatial filtering is the category to which filters that depend on the neighbourhood around the evaluated pixel belong to, as shown in Figure 2.27.

Figure 2.27: Spatial Filtering

Spatial filtering branches out into 2 categories:

- Linear Spatial Filtering, also called Spatial Convolution, where the filter has a kernel. A kernel is a matrix that will be convoluted with the image in order to filter it. An example of this type of filter is the Gaussian filter, also called Gaussian Blur.

Figure 2.28: Gaussian Kernel

Figure 2.29: Gaussian Filter Applied

- Non-Linear Filtering, where the Rank filters are contained. Rank filters are filters in which the values of the pixels in a neighbourhood are ordered by intensity. This filter will return the intensity according to the rank, meaning the pixel that will be altered can assume the maximum, the minimum or the median of the ordered list, as shown in Figure 2.30

Figure 2.30: Non-Linear Filter

Median filters are commonly used to remove a specific type of noise, called Salt and Pepper. This noise is characterized by white and black pixels randomly scattered. In a grey-scale image, since black and white are the minimum and maximum value respectively, the usefulness of the Median filter is obvious. An example of the application of the Median filter can be seen below in Figure 2.31.

Figure 2.31: Median Filter for Salt and Pepper Noise

2.6.2 Segmentation

Image segmentation is a method that generally falls in the processing phase. The purpose of segmentation is to divide an image into regions to be tagged or removed. Segmentation is commonly used for recognition and tracking. In Figure 2.32, a clear example of the potential that segmentation methods hold is shown, tagging people as red, trees as green and cars as blue. This segmentation is the result of a real-time semantic segmentation algorithm presented in [52].

Figure 2.32: Semantic Segmentation from ICNet

This, however, is not a simple method to implement and it is computationally expensive. Nevertheless, with the evolution of neural networks and advancements in parallel processing techniques, this process has even more potential for the future of computer vision [53].

Given this, it is understandable why segmentation is fundamental for applications where the manipulator is a part of the scene captured by the vision sensor. There is a need for the robot to recognize itself. For example, in obstacle detection applications the manipulator must not consider itself as an obstacle and thus must be removed from the image.

The Region Growing algorithm is an iterative approach for segmentation, where the starting pixel, or also called seed, is obtained either manually or, in certain applications, by using some detection algorithm. In each iteration, the method evaluates the neighbourhood of the seed point, and if the pixels in the neighbourhood verify an aggregation criterion, they become part of the region and future seed points for the following iterations. The aggregation criteria are normally based on a maximum threshold for pixel intensity difference between the average intensity of the region and the pixel being considered in the neighbourhood. The algorithm ends when there are no new seed points to evaluate. The neighbourhood of a pixel can be defined by the user, but the most common formats used are shown in Figure 2.33.

Figure 2.33: Common Neighbourhood Connectivity Masks

A method for segmentation that can be used for point-cloud images, which can be created from RGB-D or stereo cameras, is a 3D model-based segmentation. In [54], the manipulator is modelled as a chain of cylinders that mimic the dynamics of the real robot, given the angles of the joints. This model is then transferred into the appropriate coordinate frame and any point within the cylinders is eliminated from the point-cloud. In [55], the 3D model segmentation was optimized, through the use of the Graphics Processing Unit (GPU), for real-time systems.

2.7 Vision-Based Interaction and Obstacle Detection

Vision is a very perplexing sense but a very useful one, as most complex beings have some sort of vision sensing organ. In most animals vision provides the ability to imitate, and by imitating the perceived actions the animal eventually learns. This is why vision is important for robotics, not only to add one more sensor that can capture complex information about the environment the robot inhabits but, since the human learning skill is heavily guided by vision, it might be replicated in robots given they have access to the way humans perceive the world.

As a consequence of the miniaturization of electronics, advancements in machine learning techniques related to image processing and low-cost stereo, depth and RGB cameras, vision-based systems are currently being heavily researched and implemented. Self-driving cars, such as Tesla's Autopilot technology, implement vision all around the car, as can be seen in Figure 2.34.

Figure 2.34: Tesla's Autopilot Vision System

Integrating vision and robotic manipulators is quite common in research environments, in [56] a depth space approach for obstacle detection was presented and then applied for collision avoidance on a KUKA collaborative manipulator. This approach calculates the distances between the manipulator and obstacles directly in the depth space and is integrated with collision avoidance

through the use of repulsive vectors. According to the achieved results, while the calculated distance value is not equal to the actual Cartesian distance, the error between the calculated distance and the actual Cartesian distance is considered to be negligible. Calculating in the depth space is also proven to be significantly faster than calculating based on point-cloud information in the Cartesian 3D space. This paper belonged to a larger project called SAPHARI - Safe and Autonomous Physical Human-Aware Robot Interaction, whose overview can be seen here [57].

In [58], the Kinect v2 was evaluated and validated for mobile robot navigation applications, according to some limitations as the paper explains. In [59], a position-based visual tracking system is presented for a 7 DoF robotic manipulator, which helps simplify the effort when designing a custom controller. A real-time motion planning algorithm for a 7 DoF manipulator using a ToF camera and roadmaps was developed in [60].

Finally, in [61] a system using multiple Kinects implemented collision avoidance for a 5 DoF robotic manipulator, by actively tracking the human allowing safe physical interaction.

Chapter 3

Framework

This chapter will describe how the framework for the proposed solution was created, from a broad overview to a detailed description of the constituting parts.

3.1 System Architecture

For this thesis, a vision-based collaborative solution is proposed, in which the employed components are a Kinect v2, a UR5 manipulator from Universal Robotics, a human and a computer running MATLAB. In Figure 3.1, the main architecture of the implemented system is represented.

Figure 3.1: System Architecture

As shown above, the computer running MATLAB is the "brain" of the system, interpreting and relating visual information that is received from the Kinect v2 with information received from the UR5, and then commanding the UR5 to act upon the processed information.

MATLAB was chosen for implementation on account of its simplicity when working with matrices and the massive amount of toolboxes, functions, and support it has relative to image processing. It is also an environment that fully supports the Kinect's SDK through the use of the Image Acquisition Toolbox.

There are 3 modes implemented for the UR5:

- Self-Adaptive mode in which the robotic manipulator avoids any obstacle that may enter the defined surveillance volume;
- Collaborative mode in which the manipulator will, if possible, make contact with the user's left hand;
- Safe mode which is similar to a "return home" for mobile robots. When activated, Safe mode will simply order the robot to go to a previously defined home position.

Each of these modes is activated or deactivated by specific gestures from the user. Each of these modes, and in what manner should the user interact with them, will be described in more detail further down this Chapter.

3.2 Communications

The UR5 manipulator uses the TCP/IP protocol to communicate with exterior devices. The communication architecture for this framework is represented in Figure 3.2.

Figure 3.2: Communication Architecture

When connected, the manipulator's computer broadcasts a stream of data that it monitors internally, with the structure presented in Table 3.1:

Meaning	Type	Number of values	Size in bytes	Positions
Message Size	integer	1	4	
Time	double	1	8	1
q target	double	6	48	2 - 7
qd target	double	6	48	8 - 13
qdd target	double	6	48	14 - 19
I target	double	6	48	20 - 25
M target	double	6	48	26 - 31
q actual	double	6	48	32 - 37
qd actual	double	6	48	38 - 43
I actual	double	6	48	44 - 49
I control	double	6	48	50 - 55
Tool vector actual	double	6	48	56 - 61
TCP speed actual	double	6	48	62 - 67
TCP force	double	6	48	68 - 73
Tool vector target	double	6	48	74 - 79
TCP speed target	double	6	48	80 - 85
Digital input bits	double	1	8	86
Motor temperatures	double	6	48	87 - 92
Controller Timer	double	1	8	93
Test value	double	1	8	94
Robot Mode	double	1	8	95
Joint Modes	double	6	48	96-101
Safety Mode	double	1	8	102
Tool Accelerometer values	double	6	48	103 - 108
	double	3	24	109 - 111
Speed scaling	double	6	48	112 - 117
	double	1	8	118
	double	1	8	119
	double	1	8	120
V main	double	1	8	121
	double	1	8	122
	double	1	8	123
	double	1	8	124
V actual	double	6	48	125 - 130
TOTAL		131	1044	

Table 3.1: UR5 Real-Time Data Stream Structure

For this framework, the data of interest is contained in *q actual* and *Tool vector actual*, where *q actual* contains the joint values of the robot and *Tool vector actual* contains the Cartesian coordinates of the end-effector with respect to the base.

This data is streamed in a format that in [62] is referred to as "Matlab data". Looking at the data, shown in Figure 3.3, it seems to be Unicode Transformation Format (UTF) character encoding.

Figure 3.3: Data stream

An important side note for implementing TCP/IP communication between the UR5 and MATLAB is to remember that, when creating a *tcpip* object, this object has the *InputBufferSize* set to 512 bytes by default. This buffer must be changed to 1044 or higher, as the default size of the message the UR5 broadcasts is usually 1044 bytes.

3.2.1 Decoding Data Stream

In order to attribute meaning to the received bytes, they must be decoded. First, the UTF characters are converted into numeric bytes. In MATLAB this can be done utilizing the *unicode2native* function. The converted data should then be parsed according to Table 3.1. Since the objective is to retrieve *q actual* and *Tool vector actual* from the stream, they are found in the following positions:

$$\begin{aligned} q_{actual_{numericbytes}} &= data_{numericbytes}(253 : 300) \\ tool_{numericbytes} &= data_{numericbytes}(445 : 492) \end{aligned} \quad (3.1)$$

Each of these vectors has 48 bytes for 6 values, meaning each value is represented by 8 bytes. For the joint values, the order of the values is the following: base, shoulder, elbow, wrist 1, wrist 2, wrist3. For the tool position, the order is: x, y, z, r_x , r_y , r_z . After parsing the vectors for each variable, the bytes can now be decoded. They are then encoded to hexadecimal and reshaped.

From the reshaped hexadecimal string, the double-precision floating point number can be obtained by using the MATLAB function *hex2num*, and thus the values are decoded. A practical example for the *x* coordinate of the end-effector can be seen in Figure 3.4.


```

>> data_num = unicode2native(data);
>> tool_num = data_num(445:492);
>> x_num = tool_num(1:8)'
x_num =
    191    229     90     44     70    189     40     69
>> x_hex = dec2hex(x_num)
x_hex =
BF
E5
5A
2C
46
BD
28
45
>> x_hex = reshape(x_hex', 1, [])
x_hex =
BFE55A2C46BD2845
>> x_cartesian = hex2num(x_hex)
x_cartesian =
    -0.66726

```

Figure 3.4: Practical example in MATLAB

3.2.2 Formatting Commands

According to the UR5 scripting language manual [63], commands given or sent to the manipulator must be in a certain format so that the controller can correctly process the information. In this framework, two types of movement are supported, *movej* which is appropriate for when the motion from point to point of the end-effector does not need to be linear, and *movel*, which is used when the motion of the end-effector needs to be linear and precise.

According to what was stated previously, in order to send commands through the TCP/IP socket, it is not possible to just send a target point. This information must be formatted into the standard format, that informs the UR5 controller what type of motion is needed, the acceleration and the velocity of the motion. With the above settled, the process for formatting the data is the following:

Given a target point (x, y, z) , it must be transformed into a string with a prefix *p*, that informs the controller the vector represents a Cartesian target point, with the values between square brackets. Without this prefix, for *movej*, the vector will be interpreted as joint values to be achieved. The string that contains the point must now be placed within a larger string, with the format shown below, that instructs the controller about the type of motion, acceleration and velocity.

- *movej*: $(x, y, z) \rightarrow \text{'movej(p[x, y, z, } r_x, r_y, r_z], a = \textit{value}, v = \textit{value}, t = 0, r = 0)'$
- *movel*: $(x, y, z) \rightarrow \text{'movel(p[x, y, z, } r_x, r_y, r_z], a = \textit{value}, v = \textit{value}, t = 0, r = 0)'$

Where r_x, r_y, r_z represent the orientation of the end-effector, which is set to be neutral by the implemented formatting function, meaning $r_x = r_y = r_z = 0$.

In a *movej* type motion, a is defined as the joint acceleration, given in rad/s^2 and v is the joint speed, given in rad/s .

In a *movel* type motion, a represents the end-effector acceleration in m/s^2 and, v represents the end-effector velocity in m/s .

t is defined as the time in seconds, to perform the movement, if this value is different from 0, a and v are ignored by the UR5 controller.

r is the blend radius in meters, if this value is set to other than zero, the robot arm trajectory will be modified to avoid stopping at the point, this parameter used for continuous motion between points.

Once this string is formatted, it can be sent to the UR5 manipulator through the TCP/IP socket.

3.3 System Calibration

As mentioned in Chapter 2.4, calibration is essential when working with different frames of reference in order to be able to correctly compare data. This section will describe the process followed to obtain all the necessary parameters so that all the system's components are mathematically related.

3.3.1 Forward Kinematics

Through Forward Kinematics, the Cartesian positions of the joints can be calculated given the joint values. Following the rules of axis assignment stated in Section 2.5.1, the kinematic diagram can be constructed, resulting in Figure 3.5.

Figure 3.5: UR5 Kinematic Diagram following the DH Convention [9]

If not provided, the link lengths would have to be measured. Fortunately, Universal Robots provides the DH-parameter table for the UR5, represented in Figure 3.6, so calculating the Homogeneous Transformation matrices is trivial based on the standard stated in Equation 2.17.

UR5				
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]
Joint 1	0	0	0.089159	$\pi/2$
Joint 2	0	-0.425	0	0
Joint 3	0	-0.39225	0	0
Joint 4	0	0	0.10915	$\pi/2$
Joint 5	0	0	0.09465	$-\pi/2$
Joint 6	0	0	0.0823	0

Figure 3.6: DH parameters for UR5

Using this standard each Homogeneous Transformation matrix is calculated in MATLAB as soon as the joint values are received from the manipulator. As already stated, each Homogeneous Transformation matrix represents the rotation and translation from one frame to the frame that immediately precedes it. For example, to get the end-effector coordinates in relation to the base frame, the Homogeneous Transformation matrices must be chained in order as such:

$$P_{base} = H_{base}^{shoulder} \cdot H_{shoulder}^{elbow} \cdot H_{elbow}^{wrist1} \cdot H_{wrist1}^{wrist2} \cdot H_{wrist2}^{wrist3} \cdot H_{wrist3}^{end} \cdot P_{end} \quad (3.2)$$

Where P_{base} represents the extended coordinate vector of the end-effector's origin and P_{end} represents the extended vector that contains the origin of the end-effector's frame in relation to itself, meaning:

$$P_{end} = [0 \ 0 \ 0 \ 1]^T.$$

With this, the Forward Kinematics for the UR5 are completely defined.

3.3.1.1 DH-Parameter Error

Out of curiosity, since the UR5 sends the end-effector coordinates through TCP/IP, it would be interesting to evaluate if there is any discrepancy between the received coordinates and the calculated ones.

And so, for three random positions, the error was calculated. The error is defined as:

$$Error = Received - Calculated$$

- First position:

$$Error = (-1.5076mm \quad -0.98773mm \quad 0.31835mm)$$

- Second position:

$$Error = (-1.026mm \quad 0.03419mm \quad 0.99275mm)$$

- Third position:

$$Error = (-0.20403mm \quad -0.72376mm \quad 1.3419mm)$$

Some amount of error would be expected since the DH parameters provided are generalized for all UR5 manipulators and not for the specific one being tested. Assuming the UR5 was calibrated

at the factory, the sent coordinates should be the correct ones and will be used as such, however for this application the magnitude of the error is not very significant.

3.3.2 Intrinsic Parameters

Obtaining the intrinsic parameters of a camera is not simple since it depends on the physical construction of the device itself. Manufacturers can provide these parameters and the Kinect has a method in its C++ Application Programming Interface (API) [64], called *GetDepthCameraIntrinsics*, which provides the intrinsic parameters for the individual depth camera. However, this method seems to be defective and unsupported since its link leads to an "Unable to Service Request" message.

There are several toolboxes for camera calibration that are able to estimate the intrinsic parameters from photographs of certain patterns. Following [65], the Graphics and Media Lab (GML) C++ Camera Calibration Toolbox was selected to estimate the intrinsic parameters, since it seemed straightforward and had been tested with the Kinect v2.

Given that the depth image is created from the IR sensor, the intrinsic parameters can be obtained by taking IR pictures with the Kinect of 4 different checkerboard templates, that are provided with the toolbox. Some recommendations by the creators of the toolbox for taking the calibration photos correctly can be found in [66]. These suggestions help with positioning, as shown in Figure 3.7, the minimum number of pictures and so forth.

(Plus one more from the top of the checkerboard)

Figure 3.7: Camera Positions

With the recommendations considered, around 100 IR pictures were taken, but only 76 were deemed suitable for calibration by the GML toolbox.

The main setup for calibration can be observed in Figure 3.8, however, not all of the pictures were shot with the tripod in order to allow closer points of view.

Figure 3.8: Kinect Calibration Setup

To utilize the toolbox, it is necessary to describe the patterns used, meaning x squares by y squares, and the width, in millimetres, of the squares for each pattern, as can be seen in Figure 3.9:

Figure 3.9: GML Toolbox Interface

Feeding the images into the toolbox and running the calibration protocol returned the following matrix for the intrinsic parameters:

$$\mathbf{K} = \begin{bmatrix} 367.65 & 0 & 260.22 \\ 0 & 368.04 & 199.72 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
 f_x &= 367.65 \\
 f_y &= 368.04 \\
 c_x &= 260.22 \\
 c_y &= 199.72 \\
 s &= 0
 \end{aligned}
 \tag{3.3}$$

3.3.3 Extrinsic Parameters

The extrinsic parameters are determined by the position of the Kinect in relation to the world reference frame. For simplicity, the UR5 base frame was selected as the world frame since the end-effector's coordinates are expressed in relation to it. By choosing the robot's base frame as the world frame, and the centre of the end-effector as the interest point, calibration can be achieved by understanding where the interest point is in the Kinect's space, shown in Figure 2.15.

Figure 3.10: Relation between the Kinect frame and the World frame

To calculate the extrinsic parameters the Kabsch algorithm, (mentioned in Section 2.4.3), can be used. A MATLAB implementation of the Kabsch algorithm can be found in [40]. As input, this function takes a matrix P that contains the coordinates of the set of points in order to one frame, a matrix Q that contains the coordinates of the same set of points in order to the second frame and a m vector, which is optional, that contains the weights for each set of points. As an output, the algorithm returns the rotation matrix and the translation vector from the first frame, where the points in P are represented, to the second frame, where the points in Q are represented. The last output is the least root mean square between the two sets of points.

Given this, the sets of points must be created. This is achieved by placing the end-effector of the manipulator, which is the interest point, in a variety of positions. For each position, the coordinates in relation to the base frame must be recorded. This is quite simple since the UR5 tablet interface monitors these values, as shown in Figure 3.11. These values will constitute the input matrix P .

Figure 3.11: End-effector Coordinates in UR5 interface

For the second set of points, the process is considerably more exhaustive. Given that the intrinsic parameters are already calculated, for each position, it is required to take a picture. One would assume that a depth image would suffice, however as shown in Figure 3.12, there is no way of understanding where the centre of the end-effector resides.

Figure 3.12: Depth Image of UR5

To circumvent this issue, a marker can be placed at the centre of the end-effector and for each depth picture that is taken, an IR picture must also be shot. This way, the pixel position of the centre can be found in the IR image resorting to MATLAB:

Figure 3.13: Finding end-effector centre in IR image

Since the depth image and IR image are the same dimension and are produced by the same sensor, the pixel positions in the IR image are the exact same in the depth image, this way the depth value for a specific pixel can be extracted, once again with recourse to MATLAB:

Figure 3.14: Extracting the depth value for end-effector centre

With the pixel position and depth value extracted, by using the intrinsic parameters, the Cartesian position of the interest point in relation to the Kinect's frame can be calculated by means of Equation 3.4, which are derived from Equation 2.4. It is important to recognize that the depth value in the depth map is expressed in millimetres and is already in relation to the Z-axis of the Kinect, meaning that $z_{camera} = depth$.

$$\begin{aligned}
 z_{camera} &= depth \\
 x_{camera} &= \frac{(pixel_x - cx) \cdot z_{camera}}{fx} \\
 y_{camera} &= \frac{(pixel_y - cy) \cdot z_{camera}}{fy}
 \end{aligned} \tag{3.4}$$

These camera coordinates will constitute the input matrix Q .

Following the above-mentioned procedure, 10 end-effector positions were chosen based on two ground rules:

- The marker on the end-effector should be clearly visible in the IR image;
- The positions should be spread in space as much as possible.

Finally, the matrices were fed to the Kabsch algorithm and a Homogeneous Transformation matrix was constructed from the rotation matrix and the translation vector that were output by the algorithm.

3.3.4 Validating the parameters

To understand if the calibration had been successful, allowing the work to proceed, the parameters calculated in the previous Sections had to be validated.

As a validation test for the extrinsic and intrinsic parameters, a script would receive as input the Cartesian coordinates of the end-effector in relation to the base frame and output the depth image with the end-effector centre marked.

Figure 3.15: Output from the Validation Test

For the Forward Kinematics, it is possible to mathematically validate the Homogeneous Transformation matrices, but that would be time consuming and the UR5 does not provide information on the Cartesian position of its joints for comparison. So, it is possible to validate the Forward Kinematics visually by calculating the Cartesian position of each joint with respect to the base and then after the necessary space transformations, plotting the points onto the depth image captured by the Kinect, given that the intrinsic and extrinsic parameters are already validated, as shown in Figure 3.16.

Figure 3.16: Joint positions calculated by FK

As shown in Figure 3.15, the approximation for the intrinsic and extrinsic parameters is satisfactory but the magnitude of the error is not perceptible, and so, a secondary test was run.

The manipulator was once again put in a random position, with the marker visible. The IR image and depth image were once again taken and evaluated, resulting in:

Figure 3.17: End-effector pixel position and depth value extraction

$$pixel = (462, 132)$$

$$depth = 1747mm$$

These values were transformed into camera space. Now, with camera space coordinates calculated, the coordinates in relation to the world frame could be obtained from:

$$Coordinates_{world} = ExtrinsicMatrix^{-1} \cdot Coordinates_{camera}^T \quad (3.5)$$

Resulting in:

$$x_{calculated} = -198.23mm$$

$$y_{calculated} = -186.17mm$$

$$z_{calculated} = 928.24mm$$

The real Cartesian coordinates of the robotic manipulator were:

$$x = -197.91mm$$

$$y = -177.50mm$$

$$z = 927.00mm$$

The largest magnitude of error for this case is 8.67 mm found in the y-axis. This error is not constant in relation to the axis, meaning that for this case the y-axis had the largest error, but for other positions, this might not be true.

This magnitude and fluctuation of error is to be expected since the rotation and translation returned from the Kabsch algorithm are an approximation, based on the points fed into it. This error might be reduced by evaluating every point in the UR5's workspace and not just 10 scattered points.

3.4 Implemented Modes

This section will describe in more detail the 3 behavioural modes implemented for the UR5. Each mode has a specific gesture that will activate the requested behaviour. Only the Self-Adaptive mode requires a gesture instruction to be deactivated. When each mode is activated or deactivated, audio feedback related to the respective mode is played, so the user is aware of the behaviour the manipulator is instructed to have. In Figure 3.18 a state machine diagram explains how the system evolves according to the gestures the user is performing.

Figure 3.18: State Machine Diagram for Implemented Modes

3.4.1 Self-Adaptive Mode

As shown in Figure 3.18, Self-Adaptive mode is activated when the Kinect detects the user with both hands open. Once this gesture is detected, the computer will inform the user, through an audio message, that this mode has been engaged. This mode stays in loop until the Kinect detects the left hand of the user in the lasso position. The left hand in lasso gesture is set to break out of the loop and an audio message is issued to inform the user that the manipulator is no longer in Self-Adaptive mode. When active, Self-Adaptive mode attempts to avoid any obstacle that is contained in the considered safe volume. If the calculated target point is not within the workspace of the manipulator an audio message is played, informing the user. Out of the 3 modes implemented, Self-Adaptive is the most expensive, computationally speaking. This is a consequence of the

extensive calculations related to image processing that will be explained below. Self-Adaptive mode can be deconstructed into 3 smaller sequential segments. The main structure of this mode can be seen in Figure 3.19.

Figure 3.19: Self-Adaptive Structure Diagram

3.4.1.1 Depth Frame Processing

The objective of this segment is to identify, in the depth space, the joint and end-effector positions, meaning the pixels that represent them. The values for the joint angles and end-effector position are received from the TCP/IP socket. The Cartesian position of each joint is then calculated through Forward Kinematics. These values are transformed into camera space and subsequently into depth space. Once the pixels are obtained, the seed points for the Region Growing algorithm can be chosen.

The noise in the depth image is analogous to a type of noise called *Salt and Pepper*. For this type of noise, it is common to use a Median filter, as stated in Section 2.6.1. MATLAB has a function called *medfilt2* that implements this filter.

Returning to the segmentation algorithm, two seed points are selected in order to fully remove the manipulator. The seed points chosen were the origin of the base frame of the UR5 and the

origin of the frame of wrist 3. These particular seed points were chosen in order to avoid any error in pixel detection that might occur near the edges, meaning that, for example, if the end-effector was chosen as a seed point, in certain positions the pixel that represents the end-effector position, is one pixel away from a pixel that contains the wall behind the robot. Since the Kinect is not noise-free, even after filtering, one pixel tolerance is not acceptable and can cause the algorithm to remove the entire wall behind, since it is considered as part of the manipulator. This way, for a stationary manipulator, the segmentation algorithm removes solely the robot since the origin of the frame assigned to wrist 3 is not near any edges.

The Region Growing function for segmentation used is a MATLAB implementation by [67]. This algorithm returns a boolean matrix, where the ones represent pixel positions of the segmented area. So a custom function that employs the Region Growing function for each seed point was created. The output for each seed point is summed to the other. To find the pixel points that are to be removed, it is necessary to find the indexes of the matrix for which the result from the sum is greater than zero. These will be the indexes in the depth frame that correspond to the manipulator and are to be removed. In Figure 3.20, the process from the captured depth frame to the segmented frame is illustrated.

Figure 3.20: Depth Frame to Segmented Frame

Once the manipulator is removed from the frame, Obstacle Detection starts.

3.4.1.2 Obstacle Detection

In order to detect obstacles, a surveillance volume is created around the end-effector of the manipulator, as shown in Figure 3.21.

Figure 3.21: Surveillance Area

This virtual surveillance volume is shaped like a cube, created from a radius given in millimetres and then translated to depth space for x and y , since the values within the pixel are already in millimetres for z . Although Figure 3.21 shows a square area delimited, this only represents the pixels that are to be evaluated for obstacles. So, an obstacle is within the surveillance volume if it belongs to the square and is within the radius along the z -axis. According to [56], going from a radius, given in millimetres, to the image space is done by using the following equations:

$$\begin{aligned} x_s &= \frac{\rho \cdot f_x}{dp - \rho} \\ y_s &= \frac{\rho \cdot f_y}{dp - \rho} \end{aligned} \quad (3.6)$$

Where ρ is the radius and dp represents the depth value of the point the radius is to be designed around. The surveillance volume will be defined as:

$$SurveillanceVolume = [x_p - x_s, x_p + x_s] \cdot [y_p - y_s, y_p + y_s] \cdot [dp - \rho, dp + \rho] \quad (3.7)$$

Where x_p and y_p are the pixel points of the end-effector.

If there are no obstacles, the manipulator maintains its position. If an obstacle exists within the surveillance volume, every pixel that represents the obstacle is marked as shown in Figure 3.22 and the next segment is initiated.

Figure 3.22: Obstacle in Safe Volume Marked

3.4.1.3 Repulsive Action for Obstacle Avoidance

Once an obstacle is detected, every pixel from the obstacle is translated into camera space and subsequently into the world frame of reference. Since the end-effector's Cartesian position in relation to the world is known, for each obstacle point the Euclidean distance and direction vector to the end-effector is calculated. During the iterations, the normalized vectors are added. At the end of the iterations, the final normalized vector is divided by the number of iterations. This vector is the direction in which the robotic manipulator must move to safely retreat from the obstacle. This target point is the result of translating the end-effector position by the final normalized vector. So given a set of obstacle pixel points returned by Obstacle Avoidance:

$$obstacle_{pixel} = \begin{bmatrix} p_{xo1} & p_{yo1} & d_{o1} \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ p_{xon} & p_{yon} & d_{on} \end{bmatrix}$$

Each of the points must be transformed into camera space through Equation 3.4, and afterwards into the world frame of reference using Equation 3.5, never forgetting to extend the coordinate vector as demonstrated in Section 2.5.

Now the profile of the obstacles is represented by Cartesian coordinates in relation to the world frame, and the end-effector position is already known. By subtracting the obstacle point to the end-effector point, a direction vector is returned:

$$\vec{v}_n = \begin{bmatrix} x - o_{x_n} \\ y - o_{y_n} \\ z - o_{z_n} \end{bmatrix} \quad (3.8)$$

For each vector, the norm should be calculated:

$$\|\vec{v}_n\| = \sqrt{(x - o_{x_n})^2 + (y - o_{y_n})^2 + (z - o_{z_n})^2} \quad (3.9)$$

Then the sum of all unitary vectors is calculated:

$$\vec{r} = \sum_{i=1}^n \frac{\vec{v}_i}{\|\vec{v}_i\|} \quad (3.10)$$

$\vec{r}$ will be the repulsive vector. The target point to retreat from the obstacle will be:

$$new_{point} = (x, y, z) + \frac{\vec{r}}{N} \cdot \left(\frac{constant}{\min \|\vec{v}\|} \right) \quad (3.11)$$

Where N represents the number of iterations and *constant* is a value derived from trial-and-error in order to adjust the magnitude of the motion. The *constant* divided by the minimum Euclidean distance will set the magnitude of the repulsive vector, meaning if an obstacle is closer to the end-effector, the manipulator will move more than if the object is near the edges of the surveillance volume.

In Figure 3.19, in the Repulsive Action section, a sequence of images is illustrated where the robot backs away from the detected obstacle. Figure 3.23, tries to illustrate the formulation of the repulsive action. As it can be seen, this method takes into consideration all obstacles within the surveillance volume, commanding the manipulator to move to open space, if such space is available.

Figure 3.23: Repulsive Vector Illustration

3.4.2 Collaborative Mode

Returning to Figure 3.18, Collaborative mode is activated when the Kinect detects the right hand of the user in lasso. Once again, the respective audio message is played to inform the user that the mode has been activated. Collaborative mode does not require deactivation, once it has completed its purpose, it will return to the initial state. The Collaborative mode allows human controlled physical interaction with the manipulator. Upon activation of this mode, the position in the depth image of the user's left hand is acquired. This pixel position is then transformed into camera space, and subsequently into the world frame of reference. This becomes the Cartesian

target point for the end-effector of the UR5. Meaning this mode sends the UR5 end-effector to the left hand of the user. In case the left hand of the user is out of the manipulator's reach, an audio message is played so that the user is aware that the robot cannot make contact. This collaborative interaction can be seen in Figure 3.24.

Figure 3.24: Collaborative Mode Example

3.4.3 Safe Mode

Safe Mode is the simplest of the three. Returning one last time to Figure 3.18, it is possible to see that Safe mode is activated once the Kinect detects the right hand of the user in the closed position. Once Safe mode is activated, the respective audio message is played and the manipulator returns to a Cartesian target point that is set up *a priori*.

Chapter 4

Results and Scenarios

In this chapter, the performance of the framework developed is tested. The performance tests were conducted in the MATLAB 2016a environment with an Intel Core i5-7600K@3.8GHz and 16 GB of RAM. The user-friendly aspect of this framework will also be evaluated in order to understand if gestures are difficult to learn for users that are not familiarized with the inner workings of the system.

4.1 Code Execution Time

This section will provide an overview on the time the code takes to run for each mode.

Figure 4.1: Average Run-time Bar Graph

For the main loop, with body tracking and all modes implemented, the worst registered run-time without any mode being activated was 1.15 seconds, with its best run-time being 0.089 seconds. The average run-time was registered at around 0.805 seconds.

For the Collaborative mode, the average run-time since its activation until the mode was complete was 1.51 seconds, if the left hand of the user was in reach. If not, the mode would only run for an average of 0.012 seconds.

The average run-time for Safe mode was 1.19 seconds.

The average run-time for Self-Adaptive mode without any obstacle in the surveillance volume was around 1.36 seconds, and once an obstacle was detected, this run-time shot up to an average of 2.20 seconds, being this dependent on the number and size of the detected obstacles, which directly influences processing time.

4.2 Gesture Recognition

4.2.1 Gesture Detection Time

Gesture detection time is the time the system takes to recognize a gesture in order to activate a mode.

The gesture detection times, registered in Table 4.1, were measured from instances of the footage recorded for the scenarios

Right Hand Lasso	0.283 s	0.215 s	0.417 s
Right Hand Closed	0.995 s	0.495 s	1.014 s
Both Hands Open	0.578 s	0.365 s	0.417 s
Left Hand Lasso	0.387 s	1.05 s	0.395 s

Table 4.1: Gesture Detection Time

4.2.2 False Recognition

During the tests conducted, the Kinect did have false detections. A false detection is the activation or deactivation of a mode when the user either does not have the respective gesture showing, or has no gesture at all. Out of 145 gestures, 27 were false detections, as represented in Figure 4.2.

1	✓	21	✓	41	✓	61	✓	81	✓	101	✓	121	✓	141	✓
2	✓	22	✓	42	✓	62	✗	82	✓	102	✗	122	✓	142	✓
3	✓	23	✓	43	✓	63	✓	83	✗	103	✓	123	✗	143	✓
4	✓	24	✓	44	✓	64	✓	84	✓	104	✗	124	✓	144	✗
5	✓	25	✓	45	✓	65	✓	85	✓	105	✓	125	✓	145	✗
6	✓	26	✓	46	✓	66	✓	86	✓	106	✓	126	✓		
7	✓	27	✗	47	✓	67	✓	87	✓	107	✓	127	✓		
8	✓	28	✓	48	✓	68	✓	88	✗	108	✗	128	✓		
9	✓	29	✗	49	✗	69	✗	89	✓	109	✓	129	✓		
10	✓	30	✓	50	✓	70	✓	90	✓	110	✓	130	✓		
11	✓	31	✓	51	✓	71	✗	91	✗	111	✓	131	✓		
12	✓	32	✗	52	✗	72	✓	92	✓	112	✓	132	✓		
13	✗	33	✓	53	✓	73	✓	93	✓	113	✓	133	✗		
14	✓	34	✓	54	✓	74	✓	94	✓	114	✗	134	✓		
15	✓	35	✓	55	✗	75	✗	95	✗	115	✓	135	✓		
16	✓	36	✓	56	✓	76	✓	96	✓	116	✓	136	✓		
17	✓	37	✓	57	✓	77	✓	97	✓	117	✓	137	✗		
18	✓	38	✓	58	✓	78	✓	98	✓	118	✓	138	✗		
19	✓	39	✓	59	✓	79	✓	99	✓	119	✗	139	✗		
20	✓	40	✓	60	✓	80	✗	100	✓	120	✓	140	✓		

Figure 4.2: Correct/Incorrect Gesture Recognition

4.3 Scenarios

To evaluate the real-world application of the framework, two scenarios were created. The first scenario will be dedicated to reaction time since the correct detection of the gesture until the conclusion of the behaviour in the physical realm. The second scenario will be focused on testing the performance of Self-Adaptive mode when subjected to two obstacles, one dynamic and the other one static. The interest of this test will be to evaluate if and how long until the manipulator finds a collision safe pose.

4.3.1 First Scenario

In the first scenario, a static manipulator is being commanded by gestures from the user. The parameters of interest will be the time since the gesture is recognized until the action is complete for Collaborative and Safe mode.

4.3.1.1 First Scenario: Collaborative Mode

	Collaborative Mode Detected	Arrived at User Hand
Example 1 (4.3)	$t = 0$ s	$t = 2.487$ s
Example 2 (4.4)	$t = 0$ s	$t = 3.096$ s
Example 3 (4.5)	$t = 0$ s	$t = 3.342$ s
Example 4 (4.6)	$t = 0$ s	$t = 3.430$ s
Example 5 (4.7)	$t = 0$ s	$t = 4.208$ s

Figure 4.3: Example 1

Figure 4.4: Example 2

Figure 4.5: Example 3

Figure 4.6: Example 4

Figure 4.7: Example 5

4.3.1.2 First Scenario: Safe Mode

	Collaborative Mode Detected	Arrived at User Hand
Example 1 (4.8)	$t = 0$ s	$t = 3.280$ s
Example 2 (4.9)	$t = 0$ s	$t = 5.163$ s
Example 3 (4.10)	$t = 0$ s	$t = 3.699$ s
Example 4 (4.11)	$t = 0$ s	$t = 3.937$ s
Example 5 (4.12)	$t = 0$ s	$t = 4.938$ s

Figure 4.8: Example 1 for Safe Mode

Figure 4.9: Example 2 for Safe Mode

Figure 4.10: Example 3 for Safe Mode

Figure 4.11: Example 4 for Safe Mode

Figure 4.12: Example 5 for Safe Mode

4.3.2 Second Scenario

In the second scenario, the manipulator is placed in a position where an obstacle is already in the surveillance region, and the tester will act as the "dynamic" obstacle. During this scenario, reaction time and successful avoidance are the evaluation metrics. Within the second scenario, the *constant*, mentioned in Equation 3.11, is tested for two values: the first running at *constant* = 10 and the second running at *constant* = 30.

4.3.2.1 Running with *constant* = 10

This test took 20 moves from the manipulator, influenced by 5 "dynamic" obstacle positions and a static obstacle. The whole process took around a minute with the reaction times stated in Table 4.2. The reaction time is defined as the time since the detection of the obstacle until the finalization of the respective retreat for that input. Some instances of the process can be seen below:

Start	
1	t = 3.992 seconds
2	t = 1.833 seconds
3	t = 3.342 seconds
4	t = 1.438 seconds
5	t = 3.086 seconds
6	t = 2.645 seconds
7	t = 2.646 seconds
8	t = 2.506 seconds
9	t = 2.691 seconds
10	t = 3.04 seconds
11	t = 2.762 seconds
12	t = 2.529 seconds
13	t = 3.063 seconds
14	t = 3.04 seconds
15	t = 2.065 seconds
16	t = 4.712 seconds
17	t = 1.902 seconds
18	t = 4.666 seconds
19	t = 2.784 seconds
Finish	t = 4.689 seconds

Table 4.2: Reaction Times (const = 10)

4.3.2.2 Running with *constant* = 30

For *constant* = 30, the test also lasted around 1 minute. But in this case, there were only 7 successful movements out of 13 calculated points. The reaction time of the successful movements can be seen in Table 4.3. Some instances of the process can be seen below:

Start	
1	FAIL
2	FAIL
3	FAIL
4	FAIL
5	FAIL
6	t = 2.698 seconds
7	t = 3.992 seconds
8	t = 4.271 seconds
9	t = 2.264 seconds
NO	OBSTACLE
10	t = 3.830 seconds
11	t = 2.552seconds
NO	OBSTACLE
12	FAIL
Finish	t = 4.457 seconds

Table 4.3: Reaction Times (const = 30)

4.4 User Learning Curve

To understand if gesture activated modes for collaborative robotics are confusing to learn in real-world applications from the point of view of a person who has no knowledge on the inner workings of the framework, a small test was created. The main goal of this test is to understand if the gestures are hard to remember, based on a single explanation with a practical demonstration of the framework. This test is conducted in the following way:

A person is brought to the room where the framework is set up and is shown how the modes are activated, deactivated and their respective behaviour. After the demonstration, a small recap of the gestures and modes is done, in this recap the trigger words for the modes are set to help memorize the gestures. So the test subject should show the gesture for Collaborative mode if the tester asks for "Collaboration". When asked for "Safe Point" the test subject should show the gesture for Safe mode. To activate Self-Adaptive mode the set trigger word was "Avoidance" and to deactivate it simply "Deactivate" would be asked. The test subject is then invited to stand next to the robotic manipulator and when the tester speaks the trigger word the subject must then show the respective gesture.

The sequence of 9 gestures, described below, was tested on 20 test subjects:

- 1st Collaborative;

- 2nd Safe;
- 3rd Self-Adaptive;
- 4th Deactivation of Self-Adaptive;
- 5th Self-Adaptive;
- 6th Deactivation of Self-Adaptive;
- 7th Collaborative;
- 8th Collaborative;
- 9th Safe.

The success rate of each subject can be viewed in Table 4.4.

	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th	9 th	Score
Test Subject 1	✓	✓	✓	✓	✓	✗	✓	✓	✓	8/9
Test Subject 2	✓	✗	✓	✓	✗	✓	✓	✓	✗	6/9
Test Subject 3	✓	✓	✓	✗	✗	✗	✓	✓	✗	5/9
Test Subject 4	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 5	✓	✓	✓	✓	✓	✓	✗	✓	✓	8/9
Test Subject 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 7	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 8	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 9	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 10	✓	✓	✓	✓	✗	✓	✓	✓	✓	8/9
Test Subject 11	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 12	✓	✓	✓	✓	✓	✓	✗	✓	✓	8/9
Test Subject 13	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 14	✓	✓	✓	✓	✓	✓	✓	✓	✓	9/9
Test Subject 15	✓	✓	✓	✓	✗	✓	✓	✓	✓	8/9
Test Subject 16	✓	✗	✓	✓	✗	✗	✓	✓	✗	5/9
Test Subject 17	✓	✓	✓	✓	✗	✓	✓	✓	✓	8/9
Test Subject 18	✓	✗	✗	✓	✓	✓	✓	✓	✓	7/9
Test Subject 19	✓	✓	✗	✗	✗	✓	✓	✓	✓	6/9
Test Subject 20	✓	✓	✓	✓	✓	✓	✓	✓	✗	8/9
Overall Score:										8.7/9

Table 4.4: Score Table

Chapter 5

Discussion of Results

In this section, the results from the previous chapter and the limitations of the methods implemented will be analyzed.

5.1 System Limitations

During development, some limitations were set in order to simplify implementation:

- The system supports one user at a time;
- The manipulator is always sent to a target point with a neutral orientation, meaning $r_x = r_y = r_z = 0$;
- The system is in a controlled environment related to lighting;
- The manipulator is restricted to a workspace.

5.2 Performance

Looking at Figure 4.1, it is possible to see that as the complexity of the mode increases so does the time it takes to run, this is to be expected. The performance of the framework implemented in MATLAB serves as a proof-of-concept for a real-time implementation in C/C++. For the Collaborative mode and Safe mode, in the first scenario, the reaction times are reasonable for human interaction. The time since detection and completion of the mode feels fluid and is comfortable for the user. However, the time to completion since activation of these modes does not only depend on processing, but also on how far the manipulator is from the target point, and considering the safety of the human, the manipulator is, and always should be, set to move at low speeds.

So the general time duration for a full action could be summarized in an equation that relates the duration of the parts, as such:

$$T_{total} = T_{gesture\ recognition} + T_{processing} + T_{motion} \quad (5.1)$$

Where $T_{gesture recognition}$ and $T_{processing}$ can be enhanced with recourse to a C/C++ implementation. Nevertheless, $T_{processing}$ is also influenced by the communication velocity and computational power, meaning it can benefit from parallel programming, especially for the image processing section which relies on a considerable amount of calculations. T_{motion} will directly depend on the distance of the manipulator to the target point and its allowed maximum velocity.

5.3 Gesture Detection

Gesture Detection is a feature that the Kinect v2 employs through body tracking. The detection times in Table 4.1 are appropriate for the fluidity of the system. With clear view of the human, the Kinect is fairly capable. During the tests, the gesture that generally took longer to be detected was closed hand. This might occur since the closed hand position does not possess any protruding feature, such as lasso, where two fingers are clearly visible but not the thumb, or open hand where all fingers, including the thumb, are discernible from each other. But in closed hand, the thumb overlaps the fingers and has no distinguishable feature to lock on to. The Kinect positioning is also an important factor for correct gesture detection. If the human is not facing the Kinect, the body tracking abilities are compromised.

Looking at Figure 4.2, a small number of detections were false. This is thought to be a consequence of three things: Firstly, the moment the algorithm orders the Kinect to capture the frame; secondly, the hand position of the user, meaning the user might not intend to activate a mode but unknowingly is holding his hand in a position that the Kinect might recognize as some gesture; and thirdly, the uncertainty in the Kinect's body tracking algorithm. This problem might be solved by creating specific spatial positions for the gestures to activate the modes. As an example, the user must show the gesture and it is only valid if it is shown above the user's head.

5.4 Region Growing for Segmentation

Analyzing the Region Growing algorithm for the removal of the manipulator from the frame, it is possible to see that this method increments the complexity and the time that the Self-Adaptive mode takes to run. But this removal is still necessary and with parallel computing, this method could be enhanced even further. The main advantage of this method for segmentation is that the 3D modelling of the robotic manipulator is avoided, meaning if there are several robotic manipulators of different manufacturers on the shop-floor, this method does not need information on the physical aspect or shape of the robots, and simply receives as input the depth image and the location of two seed points along the body of the manipulator. This also allows the user to modify the manipulator with different tool-tips and add-ons without having to reprogram the segmentation algorithm.

The two main setbacks of this method are the case where an obstacle is fast enough that it is able to touch the manipulator and the one where unobstructed view to the manipulator is not possible. For the first case, once in contact with the manipulator, the algorithm will assume the obstacle is part of the manipulator and no longer an obstacle. This can be mitigated with a larger

surveillance volume or a semantic segmentation algorithm, as shown in Figure 2.32, with the Kinect's RGB camera. The second case is a bit more complicated since it is not possible to cover all blind spots with one camera. The integration of several cameras with different points of view might be able to aid in avoiding this issue.

5.5 Repulsive Vector for Collision Avoidance

In scenario 2 of the tests, repulsive action for collision avoidance is proven to be able. The difference between the *constant* being a higher value or lower value is related to the number of movements necessary, and so the time it takes to find open space. A lesser value will produce a higher number of small movements within the workspace, while a higher value will produce longer movements, and so it will more rapidly converge to the solution. But as shown in Table 4.3, this is not always true or advantageous, since a repulsive vector of larger magnitude can try to send the manipulator to space it cannot reach. This is, however, related to the positioning of the obstacles. The main advantage of this method is that there is no need for a map of the space around the manipulator or a planning phase, making it considerably useful for dynamic environments where reactive behaviour is intended. This method can be extended and enhanced by adding attractive vectors, for path recalculation, meaning target points will attract the end-effector while obstacles will repel it. A problem of this method is that it is highly dependent on the capabilities of the vision sensor, which means that if the sensor is not able to correctly capture some type of material that belongs to the obstacle, this part of the obstacle will not be accounted for, possibly causing a collision.

5.6 User Learning Curve Analysis

The last experiment, related to the human part of the process, was to understand if gestures are an easy communication manner in order to express commands. So out of 20 test subjects, the overall score for the test was 8.7 out of 9. This result is taken from a first interaction with a small demonstration, without any specialized training. From this experiment, it is possible to extrapolate that for an operator with a full day of proper training, these gestures would be simple enough to remember and use in a real-world application. The interest in having gestures for commands is that gestures remove barriers such as language, pronunciation or capturing sound from possibly loud workplaces, and according to the collected data, gestures can be considered a useful and efficient tool for human-robot communication.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

This dissertation focused on implementing a vision-based, self-adaptive, collaborative robotic system that is able to answer requests from an operator. To be able to accomplish such system the Kinect v2 was used as the vision sensor since, based on its ToF technology, it returns depth information of the captured scene. In addition to the depth information, the Kinect v2 paired with the Kinect for Windows SDK 2.0 is able to retrieve information on body tracking and gesture recognition. The robotic manipulator used was the Universal Robotics UR5 collaborative manipulator. The environment that was chosen to implement the methods that characterize the system was MATLAB, for its simplicity when working with algebraic expressions and its vast collection of toolboxes and functions related to image processing.

Calibration was shown to be an extensive process that ranged from kinematics to projective geometry. During calibration, the Kabsch algorithm, created in 1976 with the purpose of finding the best rotation matrix when comparing molecular structures, was proven to be extremely capable for extrinsic parameter estimation. This algorithm simplifies the process through which the rotation matrix and translation vector are obtained when compared with the method presented in [54].

For image processing in depth images, the Region Growing segmentation algorithm was successfully tested for manipulator recognition and removal. This algorithm is decidedly advantageous for applications as the one developed during this dissertation in comparison with the 3D model segmentation, since for Region Growing there is no need to model the manipulator in 3D. In fact, there is no need to have any kind of physical information about the manipulator such as shape and size, making this method very flexible in applications where there are several manipulators from different manufacturers or manipulators with different add-ons and/or end-effector tool-tips. This method does have flaws, the main one being, for applications that require reactive behaviour based on vision, that if an obstacle is fast enough to create contact with the manipulator, it will be considered part of the manipulator. This flaw, however, is what allows the manipulator

to have physical add-ons without any modifications of the segmentation method. For obstacle detection this issue can be countered by augmenting the surveillance volume, allowing for an earlier detection of a fast-moving obstacle, and/or adding semantic segmentation using the RGB camera of the Kinect that can differentiate between objects that are supposed to belong to the manipulator and objects that are to be considered obstacles even though they are in contact with the manipulator.

For obstacle avoidance, the application of artificial repulsive action was shown to be useful for dynamic unmapped environments. This method can be further extended to not only avoid obstacles but to do so while navigating to a target point by using artificial attractive action, effectively rendering the complete implementation of Artificial Potential Fields. The main advantage of this method for robotics in Industry 4.0 and in the future is that robots will have the necessity to react to unexpected events, and this type of reactive behaviour can be achieved through this method.

Gesture control was also shown to be an easy way to express commands to a robot, from the perspective of a human operator. Gesture control allows the elimination of language barriers and pronunciation differences. Gesture control is still highly conditioned by the quality of the tracking and recognition algorithms, but, as machine learning methods progress and computational power increases, the tracking and recognition of the human body and expressions will certainly follow suit.

With this dissertation, the developed framework demonstrates that the concept is possible for integration in CPPS, with some adjustments related to the tasks that need to be accomplished and the real-time performance. With proper training, the operators will be able to communicate with the manipulator in order to safely perform tasks in cooperation, fusing the best qualities of both humans and robotic manipulators for manufacturing lines.

The 4th Industrial Revolution has provided an opportunity to exploit vision-based collaborative robotics. These solutions will be the future for our factories, not only for mass-production industries but also for custom production lines since robots can physically collaborate with the accomplishment of a task that requires human expertise and improvisation. As soon as industries are able to implement these types of systems, the technology will translate to systems that can be integrated into other sectors. With this, the vision of robots dwelling among humans in society will not be far from sight.

The future of complex robotic systems is, however, relying on the knowledge of how biological structures function. Nature has the most complex and dynamic, highly adapted to highly adaptable beings and systems. There are a plethora of working solutions that, with further study, could be replicated in robotic systems and will, in the greater scheme, contribute to the improvement of the quality of life. Given this, a small recommendation for all who study robotics is to not ignore what already exists in the natural world. The wheel might not need to be reinvented but perhaps only reimaged.

6.2 Future Work

With respect to the framework developed during this dissertation, the future steps are evident. Firstly, the framework is to be implemented in C/C++ to allow real-time performance from the system. The next important upgrade would be to integrate attractive action with the repulsive action for navigation of the end-effector. This would increase the dynamic behaviour of the manipulator, since the robot would not only avoid obstacles but navigate through them to the target points, enhancing its value as an industrial tool. The third and final step that directly stems from this framework is to implement the Inverse Kinematics of the UR5. This would allow more control over the motion and orientation of the manipulator in order to avoid singularities. With these three steps completed, testing scenarios for collaborative tasks and shared workspace could begin.

A compelling proposal to enhance the security and reliability of the framework would be to integrate several RGB-D, or stereo cameras with different points of view of the scene to compensate for blind-spots. This could also help with the uncertainty of the body tracking algorithm since the different points of view could return reliable body tracking information after processing without depending on the position of the human. Integrating several cameras can also improve the reliability of the Region Growing segmentation, given that if one loses clear sight of the manipulator, the others will compensate and the system can continue without any major issues.

With the decline in prices of Field-Programmable Gate Arrays (FPGA) and multi-core Central Processing Units (CPU), there is a great amount of interest in optimizing and even reformulating image processing algorithms for parallel computation, effectively avoiding bottlenecks in processing performance.

References

- [1] Joao Reis, Rui Pinto, and Gil Goncalves. Human-centered application using cyber-physical production system. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017. doi:10.1109/IECON.2017.8217517.
- [2] Katie VanDyke. INDUSTRY 4.0 – WHAT IS IT? AND WHAT’S IN IT FOR ME?, 2016. URL: <http://a-cautomated.com/industry-4-0-whats/>.
- [3] Skeleton Based Action Recognition using Convolutional Neural Network. URL: <http://chetan-kumar.com/blog.html>.
- [4] Microsoft. Coordinate mapping. URL: [https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785530\(v%3Dweb.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn785530(v%3Dweb.10)).
- [5] G. Xu and Z. Zhang. *Epipolar Geometry in Stereo, Motion and Object Recognition*. 1996. doi:10.1007/978-94-015-8668-9.
- [6] What Is Camera Calibration? URL: <https://www.mathworks.com/help/vision/ug/camera-calibration.html>.
- [7] NASIM MANSUROV. What is Distortion? URL: <https://photographylife.com/what-is-distortion>.
- [8] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis and Machine Vision*, volume 91. 1993. doi:10.1007/978-1-4899-3216-7.
- [9] Ryan Keating. UR5 Inverse Kinematics. 2017.
- [10] B Camuel. Beyond Prototyping: 5 Things to Consider for Printing Your Functional 3D Parts. *Des. News (USA)*, 68(8):32 – 5, 2013. URL: <https://www.designnews.com/design-hardware-software/beyond-prototyping-5-things-consider-3d-printing-your-functional-parts/122575630432912>.
- [11] Siddhartha Kumar Khaitan and James D Mcalloy. Design Techniques and Applications of Cyber Physical Systems : A Survey. *IEEE Systems Journal*, 9:1–15, 2014. URL: https://www.academia.edu/23178627/Design_Techniques_and_Applications_of_Cyber_Physical_Systems_A_Survey.
- [12] Jay Lee, Behrad Bagheri, and Hung An Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 2015. doi:10.1016/j.mfglet.2014.12.001.

- [13] Song Han, Miao Xie, Hsiao Hwa Chen, and Yun Ling. Intrusion detection in cyber-physical systems: Techniques and challenges. *IEEE Systems Journal*, 2014. doi:[10.1109/JSYST.2013.2257594](https://doi.org/10.1109/JSYST.2013.2257594).
- [14] Ada Bagozi, Devis Bianchini, Valeria De Antonellis, Alessandro Marini, and Davide Ragazzi. Big Data Summarisation and Relevance Evaluation for Anomaly Detection in Cyber Physical Systems. In Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 429–447, Cham, 2017. Springer International Publishing.
- [15] Liliana Patricia and Saldanha Antão. Cooperative Human-Machine Interaction in Industrial Environments. 2017.
- [16] Iris Gräßler, Alexander Pöhler, and Jens Pottebaum. Creation of a Learning Factory for Cyber Physical Production Systems. In *Procedia CIRP*, 2016. doi:[10.1016/j.procir.2016.05.063](https://doi.org/10.1016/j.procir.2016.05.063).
- [17] Luis Ribeiro. Cyber-physical production systems’ design challenges. In *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017. doi:[10.1109/ISIE.2017.8001414](https://doi.org/10.1109/ISIE.2017.8001414).
- [18] *COBOTS EBOOK*. URL: <https://blog.robotiq.com/collaborative-robot-ebook>.
- [19] Michael A. Goodrich and Alan C. Schultz. Human-Robot Interaction: A Survey. *Foundations and Trends® in Human-Computer Interaction*, 2007. doi:[10.1561/11000000005](https://doi.org/10.1561/11000000005).
- [20] Kerstin Dautenhahn, Sarah Woods, Christina Kaouri, Michael L. Walters, Kheng Lee Koay, and Iain Werry. What is a robot companion - Friend, assistant or butler? In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005. doi:[10.1109/IROS.2005.1545189](https://doi.org/10.1109/IROS.2005.1545189).
- [21] Adeline Chanseau, Kerstin Dautenhahn, Kheng Lee Koay, and Maha Salem. Who is in charge? Sense of control and robot anxiety in Human-Robot Interaction. *25th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2016*, pages 743–748, 2016. doi:[10.1109/ROMAN.2016.7745202](https://doi.org/10.1109/ROMAN.2016.7745202).
- [22] Aaron Marcus. Design, user experience, and usability: Design discourse: 4th international conference, DUXU 2015 held as part of HCI international 2015 Los Angeles, CA, USA, August 2-7, 2015 proceedings, part I. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9186:590–598, 2015. doi:[10.1007/978-3-319-20886-2](https://doi.org/10.1007/978-3-319-20886-2).
- [23] Alessandra Sciutti, Martina Mara, Vincenzo Tagliasco, and Giulio Sandini. Humanizing human-robot interaction: On the importance of mutual understanding. *IEEE Technology and Society Magazine*, 2018. doi:[10.1109/MTS.2018.2795095](https://doi.org/10.1109/MTS.2018.2795095).
- [24] Jose M. Carmena. Advances in Neuroprosthetic Learning and Control. *PLoS Biology*, 11(5):1–4, 2013. doi:[10.1371/journal.pbio.1001561](https://doi.org/10.1371/journal.pbio.1001561).
- [25] SCS Concept Group. Industry 4.0: Augmented reality system for production. URL: <https://www.youtube.com/watch?v=0m67O1Em7dY>.

- [26] Volker Paelke. Augmented reality in the smart factory: Supporting workers in an industry 4.0. environment. *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014*, 2014. doi:10.1109/ETFA.2014.7005252.
- [27] Roberto Pierdicca, Emanuele Frontoni, Rama Pollini, Matteo Trani, and Lorenzo Verdini. The Use of Augmented Reality Glasses for the Application in Industry 4.0. In Lucio Tommaso De Paolis, Patrick Bourdot, and Antonio Mongelli, editors, *Augmented Reality, Virtual Reality, and Computer Graphics*, pages 389–401, Cham, 2017. Springer International Publishing.
- [28] G Avveduto, C Tanca, C Lorenzini, F Tecchia, M Carrozzino, and M Bergamasco. Safety Training Using Virtual Reality: a Comparative Approach. volume pt.I of *Augmented Reality, Virtual Reality and Computer Graphics. 4th International Conference, AVR 2017. Proceedings: LNCS*, pages 148–163, Scuola Superiore Sant’Anna, Pisa, Italy BT - Augmented Reality, Virtual Reality and Computer Graphics. 4th International Conference, AVR 2017, 12-15 June 2017, 2017. Springer International Publishing. URL: http://dx.doi.org/10.1007/978-3-319-60922-5_11, doi:10.1007/978-3-319-60922-5{_}11.
- [29] W X Quevedo, J S Sanchez, O Arteaga, M Alvarez V, V D Zambrano, C R Sanchez, and V H Andaluz. Virtual Reality System for Training in Automotive Mechanics. volume pt.I of *Augmented Reality, Virtual Reality and Computer Graphics. 4th International Conference, AVR 2017. Proceedings: LNCS*, pages 185–198, ESPE, Univ. de las Fuerzas Armadas, Sangolqui, Ecuador BT - Augmented Reality, Virtual Reality and Computer Graphics. 4th International Conference, AVR 2017, 12-15 June 2017, 2017. Springer International Publishing. URL: http://dx.doi.org/10.1007/978-3-319-60922-5_14, doi:10.1007/978-3-319-60922-5{_}14.
- [30] C Moldovan, A Craciun, V Dolga, E C Lovasz, I Maniu, and C Sticlaru. On the development of a voice and gesture based HMI for the control of a mobile robot. *Applied Mechanics and Materials*, 762:201–204, 2015. URL: <http://dx.doi.org/10.4028/www.scientific.net/AMM.762.201>, doi:10.4028/www.scientific.net/AMM.762.201.
- [31] Giovanni Saponaro, Giampiero Salvi, and Alexandre Bernardino. Robot anticipation of human intentions through continuous gesture recognition. *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, pages 218–225, 2013. doi:10.1109/CTS.2013.6567232.
- [32] Farrokh Janabi-Sharifi, Lingfeng Deng, and William J. Wilson. Comparison of basic visual servoing methods, 2011. doi:10.1109/TMECH.2010.2063710.
- [33] Microsoft. kinect hardware configuration. URL: <https://www.microsoft.com/en-us/download/details.aspx?id=44561>.
- [34] Microsoft. Kinect for Windows Programming Guide. URL: <https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn782037%28v%3Dieb.10%29>.
- [35] Joshua Blake, Florian Echtler, Christian Kerl, and Lingzhu Xiang. Libfreenect2. URL: <https://github.com/OpenKinect/libfreenect2/blob/master/README.md#maintainers>, doi:10.5281/zenodo.50641.

- [36] By Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, Richard Moore, Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. *Studies in Computational Intelligence*, 411:119–135, 2011. doi:10.1007/978-3-642-28661-2-5.
- [37] Microsoft. Coordinate Mapper Module. URL: <https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn758445%28v%3Dieb.10%29>.
- [38] WOLFGANG KABSC. A solution for the best rotation to relate two sets of vectors. (6):922–923, 1976.
- [39] W. KABSCH. A discussion of the solution for the best rotation to relate two sets of vectors. (9), 1978.
- [40] Ehud Schreiber. Kabsch Algorithm Matlab. URL: <https://www.mathworks.com/matlabcentral/fileexchange/25746-kabsch-algorithm>.
- [41] Jimmy Charnley Kromann. Kabsch Algorithm Python. URL: <https://github.com/charnley/rmsd>.
- [42] Oleg Alexandrov. Kabsch Algorithm Cpp. URL: <https://github.com/oleg-alexandrov/projects/blob/master/eigen/Kabsch.cpp>.
- [43] J.M. McCarthy. *Kinematics of Robot Manipulators*. 1987.
- [44] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. Robot modeling and control. *IEEE Control Systems*, 26(6):113–115, 2006. URL: <http://ieeexplore.ieee.org/document/4019329/>, doi:10.1109/MCS.2006.252815.
- [45] M J D Hayes and M L Husty. Singular Configurations of Wrist-Partitioned 6R Serial Robots : a Geometric Perspective for Users Configurations Singuli ‘eres des Robots S ‘eriels ‘ a Poignet Sph ‘erique et Six Couples Rot ‘oides : une Perspective G ‘eom . (March 2002), 2014. doi:10.1139/tcsme-2002-0003.
- [46] CoRoETS. What are the singularities of a typical collaborative robot (cobot), 2018. URL: <https://www.youtube.com/watch?v=6Wmw4lUHLX8>.
- [47] Andreas Aristidou and Joan Lasenby. FABRIK: A fast, iterative solver for the Inverse Kinematics problem. *Graphical Models*, 73(5):243–260, 2011. URL: <http://dx.doi.org/10.1016/j.gmod.2011.05.003>, doi:10.1016/j.gmod.2011.05.003.
- [48] Lukas Barinka and Roman Berka. Inverse Kinematics - Basic Methods. pages 1–10.
- [49] Forward Kinematics. Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 Arms. pages 1–5, 2013.
- [50] Rasmus Skovgaard Andersen. Kinematics of a UR5. pages 1–11, 2018.
- [51] Richard Scheunemann Denavit, Jacques; Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. 1955.
- [52] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. Technical report. URL: <https://github.com/hszhao/ICNet>.

- [53] James Le. How to do Semantic Segmentation using Deep learning. URL: <https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef>.
- [54] Panjawee Rakprayoon, Miti Ruchanurucks, and Ada Coundoul. Kinect-based obstacle detection for manipulator. In *2011 IEEE/SICE International Symposium on System Integration, SII 2011*, pages 68–73, 2011. doi:10.1109/SII.2011.6147421.
- [55] Blodow. Realtime URDF Filter. URL: https://github.com/blodow/realtime_urdf_filter.
- [56] Fabrizio Flacco, Torsten Kroeger, Alessandro De Luca, and Oussama Khatib. A Depth Space Approach for Evaluating Distance to Objects: with Application to Human-Robot Collision Avoidance. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 80:7–22, 2015. doi:10.1007/s10846-014-0146-2.
- [57] Università di Roma. SAPHARI Overview. URL: http://www.saphari.eu/index.php?option=com_content&view=article&id=189&Itemid=125.
- [58] Péter Fankhauser, Michael Bloesch, Diego Rodriguez, Ralf Kaestner, Marco Hutter, and Roland Siegwart. Kinect v2 for mobile robot navigation: Evaluation and modeling. *Proceedings of the 17th International Conference on Advanced Robotics, ICAR 2015*, pages 388–394, 2015. doi:10.1109/ICAR.2015.7251485.
- [59] Indrazno Siradjuddin, Laxmidhar Behera, T M McGinnity, and Sonya Coleman. A position based visual tracking system for a 7 DOF robot manipulator using a Kinect camera.
- [60] Tobias Kunz. Real-Time Motion Planning for a Robot Arm in Dynamic Environments. 2009. URL: <http://www.cc.gatech.edu/~tobias/diplomthesis.pdf>.
- [61] Carlos Morato, Krishnanand N. Kaipa, Boxuan Zhao, and Satyandra K. Gupta. Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-time Human Tracking. *Journal of Computing and Information Science in Engineering*, 14(1):011006, 2014. URL: <http://computingengineering.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4025810>, doi:10.1115/1.4025810.
- [62] Zacobria Lars Skovsgaard. Testing TCP connection Host to UR. URL: <http://www.zacobria.com/universal-robots-knowledge-base-tech-support-forum-hints-tips/knowledge-base/testing-tcp-connection-host-to-ur/>.
- [63] Universal Robots. *The URScript Programming Language*. 2013. URL: http://www.wmv-robotics.de/home_htm_files/scriptmanual_en_1.5.pdf, doi:10.1093/humrep/des191.
- [64] Microsoft. ICoordinateMapper Interface. URL: [https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn772971\(v%3Dieb.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/kinect/dn772971(v%3Dieb.10)).
- [65] Ian Ormesher. Kinect V2 Depth Camera Calibration. URL: <https://threeconstants.wordpress.com/2014/11/09/kinect-v2-depth-camera-calibration/>.
- [66] GML. Recommendations for Calibration Photos. URL: <http://graphics.cs.msu.ru/en/node/911>.

- [67] Dirk-Jan Kroon. Region Growing MATLAB. URL: <https://www.mathworks.com/matlabcentral/fileexchange/19084-region-growing>.