

Development of environmental awareness for the Kuka robot arm

M. (Mohamed) Issa

BSc Report

Committee:

Ir. F.S. Farimani

Dr. F.J. Siepel

Dr.ir. F. van der Heijden

Ir. G.A. Folkertsma

August 2016

031RAM2016

Robotics and Mechatronics

EE-Math-CS

University of Twente

P.O. Box 217

7500 AE Enschede

The Netherlands

Abstract

The Murab (MRI and Ultrasound Robotic Assisted Biopsy) project mainly aims at improving the precision of medical imaging by combining the results of MRI and US with the aid of robotic precision. The project enhances the process of screening and testing of breast cancer, not only with regard to precision, but also time consumption. This bachelor project is primarily related to the topic of safety regarding Murab. The main focus is to detect obstacles in the robot arm's environment and use an obstacle avoidance algorithm to avoid them. Xbox 360 Kinect depth sensor is used to get position information of objects in 3D space. Kuka LBR iiwa 14 R820 robot arm is to be controlled through a PC using a pre-built library. Both the sensor and the manipulator are integrated on ROS(C++). The designed system is tested and the results are analysed to evaluate its reliability.

Acknowledgements

I am thankful for the opportunity to do my bachelor assignment in RaM at the University of Twente and for the support I received from the people at the department.

Mohamed Issa
Enschede, August 2016

Contents

1	Introduction	1
1.1	Context	1
1.2	Problem statement	2
1.3	Project goal	3
1.4	Plan of approach	3
1.4.1	Requirements	3
1.5	Organization of the report	3
2	Background	4
2.1	Literature review	4
2.1.1	Birth of medical robotics	4
2.1.2	Industrial vs medical robotics	4
2.1.3	Safety guidelines for medical robotics	5
2.2	Related work	8
3	Experimental setup	9
3.1	Hardware setup	9
3.1.1	Kuka	9
3.1.2	Kinect	9
3.2	Software architecture	12
3.3	Control interface	13
4	Environmental awareness	15
4.1	Obstacle detection methods	15
4.1.1	Body detection	15
4.1.2	Depth-pixel detection	15
5	Obstacle avoidance algorithm	18
5.1	Notation definitions	18
5.2	Kinect to Kuka frame transformation	18
5.3	Forward kinematics - Kuka arm	20
5.3.1	Denavit-Hartenberg convention	20
5.3.2	Translation and Rotation method	21
5.4	Inverse kinematics - Kuka arm	22
5.5	Avoidance algorithm	24
6	Experiments and results	30
6.1	Obstacle detection and avoidance	30

6.2 Control interface	33
7 Conclusion and recommendations	36
7.1 Conclusion	36
7.2 Recommendations	36
A Appendix 1	37
A.1 control_iiwa_1.cpp	37
A.2 imitate_iiwa.cpp	39
A.3 Forward Kinematics - Denavit-Hartenberg Convention	44
A.4 Forward Kinematics - Translation and Rotation	45
A.5 Homogeneous Transformation code	46
A.6 Distance calculation code	46
A.7 Position (1 obstacle)	46
A.8 Position (2 obstacle)	47
A.9 C++ code implemented	48
Bibliography	49

List of Figures

1.1	Breast model (UMM, 2016)	1
1.2	Different imaging techniques. (a) Mammography (UMM, 2016). (b) Ultrasound (kobe, 2015). (c) MRI (kobe, 2015).	2
2.1	Robot's workspace surrounded by a cage	4
2.2	Fault tree analysis (Kazanzides, 2009)	8
3.1	Kuka LBR iiwa 14 R820 joints' specifications. (Kuka, b)	9
3.2	Kuka's safety configuration	10
3.3	Inside Kinect (Chubb, 2010)	10
3.4	Stereo Triangulation	11
3.5	Accuracy and Precision. (A) Not accurate, not precise. (B) accurate, not precise. (C) Not accurate, precise. (D) Accurate, precise (Streiner and Norman, 2006) . . .	12
3.6	ROS basic illustration	12
3.7	Joints in Kinect skeleton	13
3.8	Proposed system map	14
3.9	Extreme TM 3D Pro Logitech Joystick	14
4.1	Kinect coordinate frame	15
4.2	Kuka arm OpenCV model	16
4.3	Ros to OpenCV via CvBridge	16
5.1	Rotation from Kinect to Kuka coordinate frames	18
5.2	Translation from Kinect to Kuka coordinate frames	19
5.3	Manipulator control approaches	23
5.4	Top view illustration of model used for Inverse Kinematics. (a) Manipulator's orientation. (b) Respective model for orientation in (a).	24
5.5	Side view illustration of model used for Inverse Kinematics. (a) Manipulator's orientation. (b) Respective model for orientation in (a).	25
5.6	Distance between Point 1 and Point 2	25
5.7	Illustration of 1 obstacle avoidance method	26
5.8	Choosing factor for cartesian command	26
5.9	Illustration of 2 obstacles avoidance method	27
5.10	A plot of the factor computed against the distance calculated	27
5.11	Body Detection	28
5.12	Depth-pixel detection	29
6.1	Experiment setup	30
6.2	Body detection and obstacle avoidance in frames	30

6.3	X-axis plots. (a)Right hand position in x-direction. (b)End effector position in x-direction	31
6.4	Y-axis plots. (a)Right hand position in y-direction. (b)End effector position in y-direction	31
6.5	Z-axis plots. (a)Right hand position in z-direction. (b)End effector position in z-direction	32
6.6	Joystick axes angles. (a)x-axis (b)y-axis (c)z-axis.	33
6.7	End effector X, Y and Z position.	34
6.8	Robot arm copying human's arm.	34
6.9	Human arm controlling 4 of the manipulator's joints	35

List of Tables

2.1	Failure modes effects analysis sample (Kazanzides, 2009)	7
5.1	Table of Notation My Research	18
5.2	Denavit-Hartenberg Parameters	20

Foreword

Reaching the ultimate, most secure and safe robotic environment would require adding human-like senses to the robot. This offers constant monitoring of the aspects of the environment at real-time and reacting to all scenarios according to a predefined strategy. Think about it! What if your system could hear? See? Think? Or even Talk to you? This is definitely where science, technology and research are heading. In this project, the rock of vision was scratched, striving to reach the goal of having an environment-aware robot.

1 Introduction

1.1 Context

It is necessary -for understanding this project completely- to give a glimpse of the Murab project, the importance of safety for any medical procedure and to show how each part is correlated with the other to give the best possible end-product. The Murab (MRI Ultrasound Robotic Assisted Biopsy) project mainly aims at improving the precision of medical imaging by combining the results of MRI and US with the aid of robotic precision. The project will enhance the process of screening and testing of breast cancer, not only with regard to precision, but also time consumption.(Agreement, 2015) Like any other project, safety is considered a major aspect that has to be dealt with professionally. Not only for including a robotic arm, but also for being a medical project that deals and constantly interacts with doctors and patients at all times.

Breast cancer starts off from the breast cells and can be present on either the interiors of the milk ducts or the tiny lobes that supply the milk as illustrated in Figure 1.1. Either way, a tumor growing can be a benign or a malignant one. Benign tumors tend to be harmless and they don't replicate to other parts of the body. On the other hand, malignant tumors are cancerous and can infect other parts of the body and grow into neighboring organs.(AmericanCancerSociety, 2014) Women and men -mostly women- have been victims of breast cancer since the earliest case discovered in ancient Egypt 1600 BC.(Brechon, 2013) Surprisingly, breast cancer is the most prevailing cancer in the world for women. 25% of cancers in women in 2012 were breast cancer with 1.7 million new cases discovered this year worldwide.(IARC, 2014) If these numbers tell us something, it's that the problem is definitely severe and that crucial steps and actions of research must be taken as soon as possible.

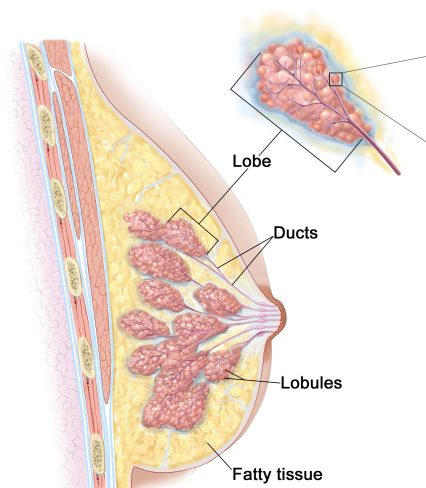


Figure 1.1: Breast model (UMM, 2016)

Usually breast cancer awareness campaigns thrive for nothing more than seeing people being able to identify if they see symptoms of breast cancer in their body. Breast lumps, breast and nipple pain, swelling of the breast or just breast skin irritation might be a call to visit the doctor for a check-up.(BREASTCANCER.ORG, 2014) As explained by Radiologist Dr. J. Veltman, from ZGT hospital, usually after doctors examine the patients, they choose to go with one of the imaging options either a mammography, US or MRI. If the reader is not familiar with the appearance of any of those, refer to Figure 1.2. In US, if it's visible, doctors can identify whether the lump is a cyst or not. Cysts are harmless sacs containing fluid that grow in breast tissues

and their removal is not crucial except if they are causing discomfort for the patient. If the lump is not visible in US, further MRI is to be done for doctors to have a full image of the lump and precisely have information on its position. It is often left for the last possible type of imaging to go through as it is the most expensive and at the same time patients need to lay motionless for a while in a closed machine which is quite uncomfortable for claustrophobics. After detecting the lump and having a reliable image and an accurate position of it, radiologists would make a breast biopsy either US or MRI guided according to the method used for image detecting. For a comparison between both types of biopsies, it is as simple as that US guided biopsy is widely available, low cost and it offers real time imaging of the needle. On the other hand, although MRI screening in general offers a more clear and detailed image of captured part of the body, MRI guided biopsy is not preferred because of its extremely high cost and the discomfort it causes to the patients. That is mainly why doctors strive to target-US all the MRI screenings to perform an US guided biopsy. Unfortunately, only 50% of those MRI screenings succeed in being targeted on US leaving the other 50% with no option than performing an MRI guided biopsy. Murab basically combines the best of MRI which is its extremely high precision and sensitivity and the best of US which is its compatibleness and practicality. The image taken from the MRI is blended and integrated with the US image in a computer-assisted image fusion which is also more reliable than the user-dependent targeted US. MURAB uses a KUKA robotic arm along with an US probe equipped with a needle as its end-effector. The needle is to be positioned in front of the lesion using the image generated and the doctor is to insert it for the biopsy to finally take place.

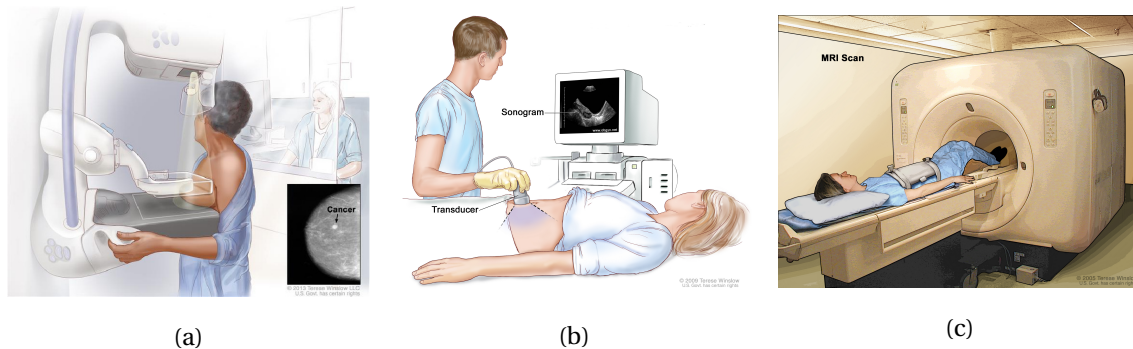


Figure 1.2: Different imaging techniques. (a) Mammography (UMM, 2016). (b) Ultrasound (kobe, 2015). (c) MRI (kobe, 2015).

1.2 Problem statement

When robotic arms are mentioned, some keywords almost always pop up in one's head. One of those keywords is definitely safety, especially when the robotic arm is related to not only human interaction but also medical procedures. Safety is quite a huge word that can never be looked at from a single aspect. For the sake of this project, viewing safety will be discussed in relation to the Murab project in a way that the robot arm's environment should be under constant screening. The optimum way to reach -what we call- a safe working environment, is to make the arm fully aware of its static and moving boundaries. Static boundaries are basically defined by the positions and orientations for the arm that are and will always be off-limits. Meaning that no matter what level the procedure is at and no matter where the doctor or the patient are positioned relative to the arm, the static boundaries will always be constants and must always be avoided. Static boundaries are briefly listed as the mechanical constraints for the robot arm which basically defines its workspace. Not only that, but also if the environment of the robot arm is known, any static, non-moving objects are considered to be static boundaries. On the

other hand, moving boundaries which will obviously be harder to track and prevent can be listed as any changeable and unstable object that lies in the arm's workspace. The doctor and the patient will definitely be the first thing that comes to mind when dealing with unfixed objects, although eventually non-humans must be accountable as safety constraints as well.

1.3 Project goal

This project's goal is summarized in two points. Firstly, a thorough study on safety guidelines for medical robotics should be prepared. Secondly, there should exist a state where obstacles around the arm are detected and the arm is controlled in a way to avoid these obstacles. In other words, this second point can be simply stated as firstly detecting obstacles around the arm and secondly avoiding them using an obstacle avoidance algorithm.

1.4 Plan of approach

To begin with, a camera or more precisely a vision sensor must be used to account for the inadequacy or lack of vision sensing in robots. A depth sensor would be extremely crucial as well so as to accurately receive the position of the object detected relative to the sensor. Moreover, controlling the robot arm using a stand-alone PC and integrating both the arm's control with the depth sensor information will make the safety system applicable.

1.4.1 Requirements

For any system to be constructed, the requirements of the end product should be listed and thoroughly discussed.

1. Safety for medical robotics study.
2. Detecting obstacles in the arm's environment.
3. Avoiding obstacles detected.
4. Xbox 360 Kinect sensor and Kuka robot arm were required to be used.
5. Clear descriptive *Readme* files for using the system.

1.5 Organization of the report

In chapter 2, there is a detailed literature study on safety for medical robots. Afterwards, in chapter 3, the hardware and software used in the project are introduced to give the reader enough background on what the project is made of. In chapter 4, two methods for environmental awareness of the robot arm are proposed. Furthermore, chapter 5 focuses on methods of avoiding the obstacles in the arm's environment. Chapter 6 consists of the experiments, results and assessments of the system designed. Finally, chapter 7 will conclude the report and a number of future recommendations will be mentioned.

2 Background

2.1 Literature review

2.1.1 Birth of medical robotics

Surgical robotics have proved popularity only 10 years after it was first introduced in 1985.(Kazanzides, 2009; Gomes, 2011). The use of robots in surgery is highly controlled by the da Vinci system of Intuitive Surgical (Sunnyvale, CA, USA),especially in minimally invasive surgery,(Guthart and Salisbury Jr, 2000) although lots of other robotics manufacturers are striving to enter the market.(Gomes, 2011)

2.1.2 Industrial vs medical robotics

The popularity of medical robotics is widely vital for several reasons. Geometric accuracy, force precision and immunity against fatigue are main advantages of the machine over human in general as seen by Duchemin et al.(Duchemin et al., 2004)

On the other hand, robots are not flawless, as they obviously lack having the ability to take decisions or adapt to new environments.(Duchemin et al., 2004). Dealing with these drawbacks for medical robotics as dealt with those of industrial robots is fundamentally impossible. In industrial robots, accidents mainly vary from collision accidents to crushing or trapping of worker's limbs or even mechanical part accidents in which certain links/motors mechanically fail.(OSHA, 2002). Humans are usually safe against these kinds of hazards by managing a workspace for the robot and disallowing entering this workspace during the robot's operation as shown in Figure 2.1. Proof of effectiveness of such method lies simply in having many industrial robot accidents occur only during programming, maintenance or adjustment and not during operation.(OSHA, 2002). As far as direct contact with the robot during operation is avoided, safety is maintained.



Figure 2.1: Robot's workspace surrounded by a cage

Quite the contrary, medical robotics need more and more guidelines for operation as their main working condition is directly contacting the patient at all times. Nevertheless, different sizes, characteristics and position of the patients' body highly affect the process. Sterilization of the robot and its end-effector is a must at all times, in addition to the robot being highly mobile for the need to transport it to or from an operating theater. Furthermore, there exists a variety

of surgical robotic systems; passive (no actuators included), semi-active (actuators used as a guide to surgeons) and active systems (fully-automatic actuated joints). (Duchemin et al., 2004).

For the sake of comparison, going back to industrial robotics, several sources of danger are foreseen to have the probability of happening. From simple human control/programming errors to unauthorized access to the robot's work envelope, and from internal mechanical faults to main power system failures are thought to be expected hazards. (OSHA, 2002). Proposed approaches to be considered for safeguarding include risk analysis as well as constant maintenance of the robot. Moreover, having devices that alert the operator for errors/faults in the system is recommended. Operators in general are required to have passed a safety training before handling direct contact with the robot. Finally, having safeguarding devices which basically limit and control the motion of the robot in its working space, is favoured as well. (OSHA, 2002).

2.1.3 Safety guidelines for medical robotics

Shifting to the main topic -safety for medical robotics-, it has been argued that perfecting a safe environment for a medical robot is not an effortless, doable thing. There has been quite controversy in coming up with the regular safety guidelines for all medical robotics, due to the reasons mentioned above. For the impossibility to mention all the debates/discussions, I will refer to the most widely applicable and general to relate to Murab project.

Hardware design

To begin with, (Kazanzides et al., 2008) have argued that steps towards a safe medical robot include sensing both internally and externally. Internal sensors include encoders mounted on joints to measure the angle of rotation of links. On the other hand, external sensors would be trying to perfect human senses. Force sensors and vision systems are believed to be quite an optimal option. Moreover, geometric relationship between patient's anatomy, robots and sensors must be known at all times. Nevertheless, also (Wang et al., 2006) agrees that having a powerful user-friendly user interface is an adequately crucial option. For instance (Kazanzides et al., 2008) mention that foot pedals are one of the finest options for interacting with the robot since it needs no sterilization, in addition to it not interfering with doctors' hands.

Secondly, as advised by (Wang et al., 2006), a safe medical robotic system should be developed by the integration of expertise of electrical, mechanical and software engineers, along with surgeons and physicians. In (Duchemin et al., 2004), Gilles Duchemin et al think highly of intrinsic safety as they judge it is best to limit the robot's actuators' power as needed in application instead of referring to software threshold limits using programming. Moreover, mechanical torque limiters can be added to joints and using high reduction gears can help limit the manipulator's velocity. Furthermore, for applications where a force is acted on a human being -like most surgical applications-, it is necessary to use a mechanical system for detaching the end effector in case of robot's failure. Last but not least, in emergency stops, having reliable mechanical brakes for the joints is crucial. (Duchemin et al., 2004) still believe that it is not the best option to go with at all times as robots shiver quite a bit when brakes are applied, so as an substitute applying an equal force in the opposite direction tends to act as a gravity compensator for the system.

Additionally, having redundant sensors in the system is an attractive approach to aid against hazards caused by sensors' failure, as seen in (Kazanzides et al., 2008; Kazanzides, 2009). Using two parallel sensors and assuring that readings from both sensors fall in the same threshold guarantees the system is operating correctly. Otherwise, if one of the sensors fails, then the error between both readings will exceed the threshold. This approach works correctly as mentioned in (Kazanzides, 2009) when two parallel encoders placed on the joint which eliminates single point of failure. Furthermore, Peter Kazanzides et al. argues that wrong implementation of choosing the position of sensors might not eliminate single point of failure after all. More-

over, using redundant tools might not be a wise choice as well. For instance, in a system of parallel pneumatic tools, tracing of failure of one of them is impossible, since the system will be working normally. On the contrast, Gilles Duchemin et al.(Duchemin et al., 2004) believe that the use of redundancy in the system should be analyzed as it definitely increases the system's complexity and cost and it is inversely proportional to the system's reliability.

Moreover, to be more directed towards Murab, mechanical details of the surgical robot arm should include some necessities as seen by Gilles Duchemin et al.(Duchemin et al., 2004). Firstly, all electric cables should be placed inside the core of the arm. Besides, having mechanical joint limits is preferable to have a well-known working space. All the links' dimensions should be known so that the "safe" surrounding of the robot could be precisely known. Moreover, singularities of the wrist and shoulder must be avoided at all times. This is done in the (Pierrot et al., 1999) Hippocrate project by stopping the motion of the robot when it reaches a singularity, and the arm is moved away manually by the operator. Furthermore, Gilles Duchemin et al.(Duchemin et al., 2004) think it is an absolute necessity to have a dead man switch (DMS) in the system, which basically gives a signal to the robot allowing it to operate as long as it is pushed. As soon as the DMS is released the robot comes to rest immediately. Inspired by Peter Kazanzides et al.(Kazanzides et al., 2008), this can be used in the form of a pedal to avoid sterilization problems.

Software design

Furthermore, Software requirements are absolutely tremendous and they should be. Regardless of the decrease in profitability due to its high cost, real time controllers are absolutely crucial to the system as mentioned by Yulun Wang et al.(Wang et al., 2006). A quite detailed approach of building a reliable and powerful software is demonstrated in (Duchemin et al., 2004). Gilles Duchemin et al. explain that the controller should prioritize tasks with security at the top of the priority list. Not only a backup of the system is recommended, but actually having a separate CPU for each "primitive function" is favoured. Additionally, adding redundant joint position, velocity and torque limits, less than the mechanical limits would be beneficial. It would increase the lifetime of the arm's mechanical parts and would increase the safety of the system.

Gilles Duchemin et al.(Duchemin et al., 2004; Kazanzides, 2009) see that having watchdog as a part of implementing a safety loop allows constant check-up of the software's working condition. The watchdog is an independent, external hardware device which disables power to the main system's actuators in case of processor's failure detected.

Redundancy

As mentioned previously, having redundant sensors will check for system failure, so if the difference between the two readings -known as the error- exceeds a certain threshold, then a system failure has took place. This calls for the control software to disable power reaching the actuator, via a simple relay circuit. Peter Kazanzides et al.(Kazanzides et al., 2008) shows that the maximum joint position that can be reached is the sum of the error maximum threshold, the maximum joint velocity multiplied by the control period and the distance travelled by the robot after power is turned off, due to inertial or external forces.(image). This equation shows that limiting/decreasing the maximum velocity, limits/decreases the maximum joint position. It is also recommended in (Kazanzides, 2009) to set the error threshold between the measured position and the commanded position to be no less than the maximum incremental command to the desired position. As a result, this threshold can reach a maximum which reduces the safety effectiveness of the system. Therefore, another approach is to change the threshold according to the status of operation the robot is in. For example, whether the robot is stagnant or in motion with high or low velocity.

Risk analysis

The system's reaction to failures should be thoroughly studied and considered, since it is not sensible to react to harmless sensor faults like extreme system power failures. Therefore several safety analysis methods are proposed in (Kazanzides et al., 2008; Duchemin et al., 2004; Kazanzides, 2009; Fei et al., 2001). The most popular method of analysis is the Failure Modes Effects Analysis (FMEA), in which could be defined as an ascending form of analysis where possible failures of specific components of the system are studied and their effect on the system generally is investigated.(Kazanzides et al., 2008)). This method is covered by (fme, 2006) and is recommended in (Kazanzides et al., 2008; Duchemin et al., 2004; Kazanzides, 2009). A sample of the FMEA report is illustrated in table 2.1. Peter Kazanzides et al. in (Kazanzides et al., 2008) also proposes a quantitative way of making analysis which adds criticality to the name of the method making it Failure Modes Effects and Criticality Analysis (FMECA). This consists of a Risk Priority Number (RPN) which is basically computed by the multiplication of the severity (S), the occurrence (O) and the detectability (D) of the failure to take place. Inspired by the Failure Care Taking in (Duchemin et al., 2004), the (RPN) can be used to identify the robot's suitable action to be taken according to the event that is to happen. For instance, whether the robot should decelerate until it reaches total immobility, the robot's motion should be paused until some error is fixed by the operator or maybe even immediate power should be cut and mechanical brakes applied.

Failure Mode	Effect on System	Cause	Method of Control
Incorrect feedback	Incorrect robot motion	Encoder failure	Redundant encoders with software check
Uncontrolled motor current	Incorrect robot motion	Power amplifier failure	Tracking error software check
Robot continues previous motion	Incorrect robot motion	Processor failure	Watchdog to disable power

Table 2.1: Failure modes effects analysis sample (Kazanzides, 2009)

Another very popular approach for risk assessment is called Fault Tree Analysis (FTA), as standardized in which is roughly the opposite of FMEA or FMECA. It is standardized in (fta, 2006) and is basically a descending form of analysis where the main system failure itself is traced to the original faulty component that caused this failure. FTA is recommended to be illustrated graphically and is mostly beneficial in analysing the crisis after happening. The corresponding FTA to the sample FMEA shown previously is illustrated below in 2.2. Finally, Baowei Fei et al. in (Fei et al., 2001) proposed a safety model than analyses the system for medical robots which is called Hazard Identification and Safety Insurance Control (HISIC). It is argued that errors in the system can occur due to human error or system error which can either be totally Hardware errors, totally Software errors, Hardware errors generated by Software or Software errors generated by Hardware. System safety analysis or as mentioned in (Fei et al., 2001) Safety index is estimated to be $f(\text{SW}(\text{PL}), \text{HW}(\text{PL}))$ where SW reflects the value of Software factor, HW reflects the value of Hardware factor and PL reflects the value of policy factor. Besides recommending FTA as a praised safety analysis method, (Fei et al., 2001) managed to dig deep into some identification methods for hazards to the system and also to define some approaches to limit, monitor and control medical robotics in general.

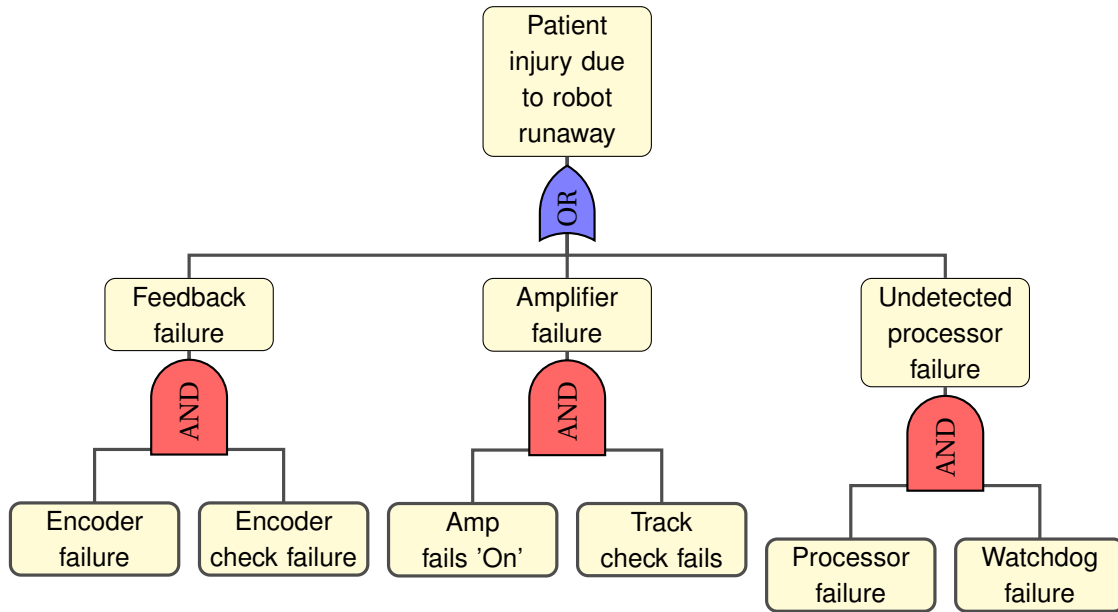


Figure 2.2: Fault tree analysis (Kazanzides, 2009)

Future work overview

To end with, as analysed in (Gomes, 2011), tiny, low-priced, economical robots are defining the future. Engineers and research are aiming at looking more towards designing micro and nano devices which can be swallowed by the patient and afterwards identify and treat specific tissues. As concluded in (Kazanzides, 2009), the increase variety and diversity in medical robots, as well as physical variations in human, both make it a close to impossible mission to develop general safety guidelines for medical robots. Also, in (Kazanzides et al., 2008), Peter Kazanzides et al. believe that as crucial as validating the system is, it is almost impossible to do such thing, due to the fact that it is unobtainable to simulate real-life clinical conditions. More than one perspective in the previous literature relied on the latest guidelines for safety for medical robotics in Europe (ISO, 2012) which includes (EEC, 1993), (EEC, 1998) and (EEC, 1990). **i.e.** These include both FMEA and FTA analysis approaches.

2.2 Related work

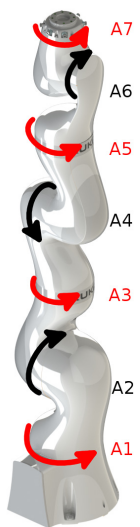
Several previous work has tried to link depth sensors to robots in general. In (Kuhn and Henrich, 2007), a depth sensor has been used to calculate the approximate distance between some known and some unknown objects. (Biswas and Veloso, 2012; Maier et al., 2013; Chen et al., 2014; Wang et al., 2014). Kinect has been used often in certain kinds of applications where depth information is needed. (Jamaluddin, 2014). Furthermore, a thorough study has been implemented to facilitate and test the use of Kinect with robotic systems (El-laithy et al., 2012). Obstacle Avoidance is a crucial pillar of autonomous robotic control, as seen in (Seraji et al., 1997; Zohaib et al., 2013), where more than one approach is studied. Collaborating Kinect with robotic arms for applying collision avoidance has been implemented in (Vachálek et al., 2015; Ueki et al., 2015). Moreover, Kuka is becoming a crucial member in the medical robotics field these days. (Kuka, 2013). In (Ogrinc et al., 2011; Flacco et al., 2012; Comparetti, 2014), Kinect and Kuka LWR 4+ were used together to perform obstacle avoidance for the manipulator. Although many aspects of safety has been discussed in this chapter, I believe focusing on one main requirement in this report is crucial for getting the most adequate results. Therefore, in this report, obstacle avoidance mainly will be tested for Kuka iiwa 14 R820 model using Xbox 360 Kinect Sensor.

3 Experimental setup

3.1 Hardware setup

3.1.1 Kuka

To begin with, the robot arm manipulator used in the project was a KUKA LBR iiwa 14 R820. LBR basically stands for "Leichtbauroboter" which means "lightweight robot" in German and "iiwa" stands for "intelligent industrial work assistant". This light-weight robot weighs only 29.9 kg but offers payload up-to 14 kg. It is considered a redundant manipulator with 7 rotational joints which form 7 axes. (Kuka, a,b). The dimensions, the workspace and speed and torque figures for each joint of the manipulator are shown in Figure 3.1.



	Range of Motion	Maximum Torque	Maximum Speed
Axis 7 (A7)	$+/- 175^\circ$	$40Nm$	$135^\circ/s$
Axis 6 (A6)	$+/- 120^\circ$	$40Nm$	$135^\circ/s$
Axis 5 (A5)	$+/- 170^\circ$	$110Nm$	$130^\circ/s$
Axis 4 (A4)	$+/- 120^\circ$	$176Nm$	$75^\circ/s$
Axis 3 (A3)	$+/- 170^\circ$	$176Nm$	$100^\circ/s$
Axis 2 (A2)	$+/- 120^\circ$	$320Nm$	$85^\circ/s$
Axis 1 (A1)	$+/- 170^\circ$	$320Nm$	$85^\circ/s$

Figure 3.1: Kuka LBR iiwa 14 R820 joints' specifications. (Kuka, b)

Although the Kuka robot arm already has factory-set configurations installed(Kuka, a), further safety configurations were made for several reasons. To begin with, safety configuration usually depends on the project's environment, workspace and whether or not there will be direct contact with human-beings. Moreover, having project-related safety configurations overwrites the arm's initial safety configurations so that they are never reached. Reaching the predefined configuration's limits automatically causes the arm's joints to lock on its current position and stops immediately. On the other hand, the response to reaching the project's configuration's limits is totally controllable and was chosen to be immediate stop of the arm at its current position and pausing the program running. An example of Kuka's safety configuration layout is seen in 3.2.

3.1.2 Kinect

Kinect is a motion sensing device introduced by Microsoft primarily for Microsoft's Xbox gaming consoles late 2010.(Kinect, 2009) Later in 2012, another version of Kinect was introduced to be Windows compatible. Although some of the specifications and technology used regarding Kinect are available as referenced in the official Microsoft website, some information are not entirely available. Thus, some information are referenced and gathered through the efforts of

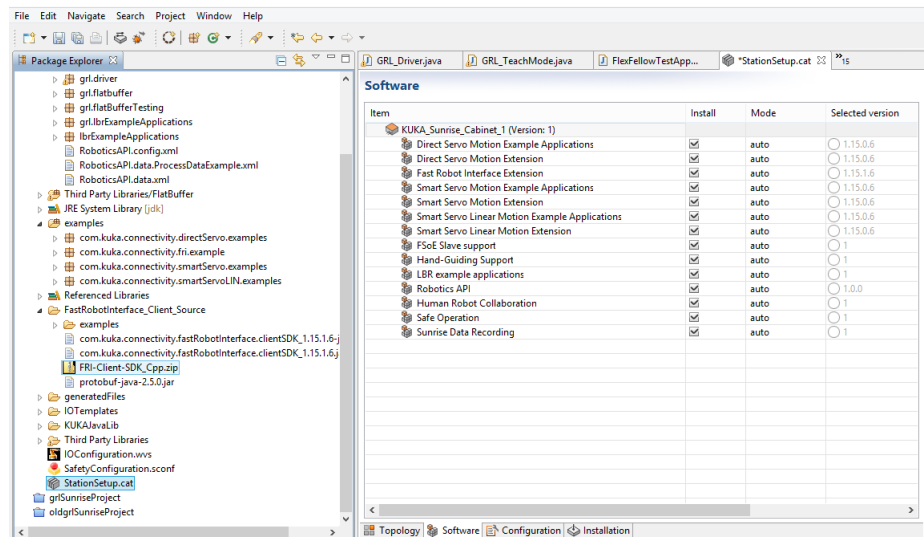


Figure 3.2: Kuka's safety configuration

individuals through what is known as reverse engineering. Furthermore, a detailed overview of the specifications of the Kinect Version1 model: 1414 which will be used in this project will be introduced in the following section.

General Specifications

Kinect is basically a combination of not only a depth sensor but also an RGB camera. The RGB camera outputs video of the three basic color components at a frame rate of up to 30 frames per second. It offers resolution of 640 x 480 pixels at 30 frames per second, although it is mentioned that its resolution can reach 1280 x 1024 pixel at much less frame rate of course. (Microsoft, 2012). More importantly, for depth sensing, Kinect contains an infrared projector along with a monochrome CMOS sensor to get depth information using a technology that Primesense - the producer of Kinect's depth sensor- refer to as "light coding". (mirror2image, 2010) Depth sensing is offered at a resolution of 640 x 480 at 30 frames per second as well. (Microsoft, 2012)

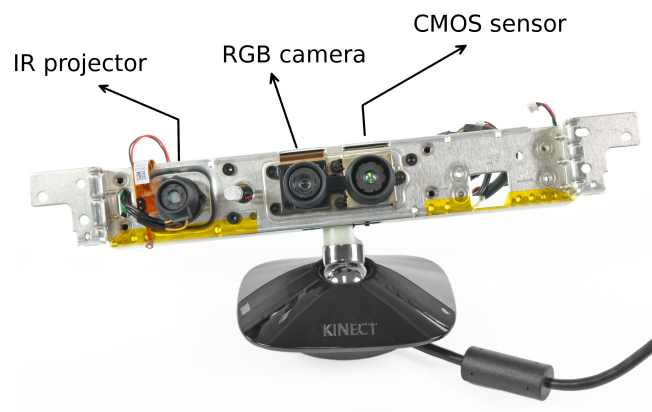


Figure 3.3: Inside Kinect (Chubb, 2010)

Depth Measurement

Depth measurement is the most related and most crucial topic to the project. The technology, as mentioned before is called "light coding", uses an IR projector that projects what is known as pseudo-random points of IR light to code the scene. The light returned is read using the CMOS sensor which is distorted in a way that reflects the depth of objects in the scene. Furthermore, for the depth of each pixel to be calculated, the distortion is used in a process known as Stereo Triangulation as seen in 3.4. An image from the IR sensor is used along with another image which is already known and hard-coded in the chip logic since the projected IR is a set of pseudo-random points which is really similar to structured light. Finally, Stereo Triangulation uses both images after calculating the horizontal offset to get the depth information of the pixel measured. (mirror2image, 2010). The Kinect depth sensor limits depth information values from 800 mm to 4000 mm. The angular field of view provided is 57° horizontally and 43° vertically. (Microsoft, 2012)

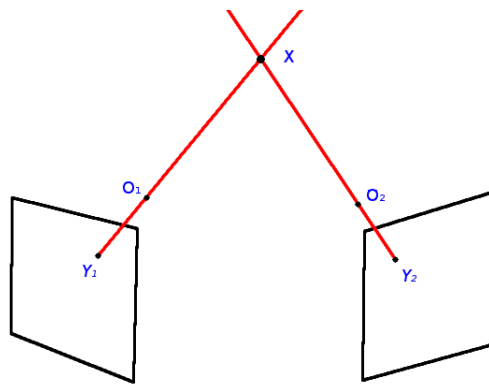


Figure 3.4: Stereo Triangulation

Reliability assessment

According to (OpenNI and ROS, 2011), Kinect depth information are accurate to ± 1 mm. To begin with, it is quite necessary to define some critical terms that are almost always linked to safety measures. The following terms explained by (BIPM et al., 2008) will be used in testing the results of the approaches introduced further in the report.

1. Accuracy: This term reflects how close the measured value is to the true value.
2. Precision: This term reflects how close the measured values from repeated measurements on the same object under the same conditions are.
3. Verification: Supplying unbiased evidence that a certain item fulfils certain requirements.
4. Validation: Verifying that the certain requirements are sufficient for a proposed use.
5. Reliability: This term basically reflects the consistency of the positive performance of a certain system.

Some publications like (Streiner and Norman, 2006) deal with accuracy and validations as synonyms and precision and reliability as synonyms as well. For the sake of this report, accuracy and precision will be dealing with measured test results and validation and reliability will be linked with the system's general evaluation and comparison to initial requirements. Reliability will also be linked to system's results' consistency with multiple users.

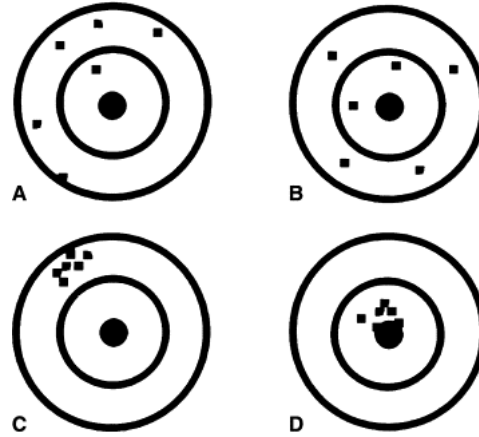


Figure 3.5: Accuracy and Precision. (A) Not accurate, not precise. (B) accurate, not precise. (C) Not accurate, precise. (D) Accurate, precise (Streiner and Norman, 2006)

3.2 Software architecture

To begin with, all the software I have used are free and open-source with a license that allows the user to publish the work for commercial purposes. It should be cross-platform as well, so as not to find difficulty when integrating different parts of the project with other team members. Operating system used in the whole project was Linux Ubuntu 14.04.4 LTS. All software/code for low level implementation were written in C++ language except for mathematical simulations where maxima language was used in **WxMaxima**.

ROS

The **Robot Operating System**, also known as ROS, is an open-source, meta-operating system which offers a well-arranged communications platform. The basic idea behind ROS is its simplicity. Any *node* can publish *msgs* to a *topic*, and on the other side any other *node* can subscribe to any *msgs* on a *topic*. (Quigley et al., 2009). ROS was used along with its commands and services, integrated with C++ language for all source and launch files. This is illustrated thoroughly in the following Figure 3.6 where */joy_node* is publishing to */joy* and */iiwa/control_iiwa_1* is subscribing to it.

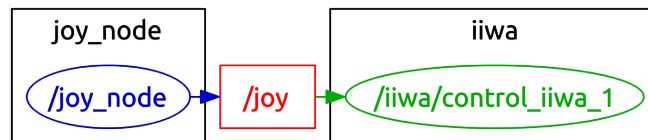


Figure 3.6: ROS basic illustration

Kuka

Despite the availability of Kuka's new user-friendly platform for controlling the iiwa model (Kuka, a), we could not really make use of it in this project for more than one reason. To begin with, it is java-based and only available on Windows operating system which basically opposes two of the main rules for the project that all software/code be implemented in C++ on Ubuntu only. Fortunately, an open-source, BSD licensed software stack, **iiwa_stack** was used to implement the communication between ROS Packages and the Kuka controller. (Virga and Esposito, 2015).

Kinect

Since the official SDK by Microsoft is only supported in Windows, an alternative had to be found to be applicable in Linux. Fortunately, Primesense, the company behind manufacturing the depth sensors for Kinect, which has been acquired by Apple, was a founding member of an open-source software project called OpenNI. OpenNI was basically responsible for reading the sensor depth and RGB data from Kinect and sending them to my system. Furthermore, Primesense's motion tracking middleware "NITE", was quite crucial in the project for its advantageous gesture and skeleton tracking.(Mitchell, 2010)

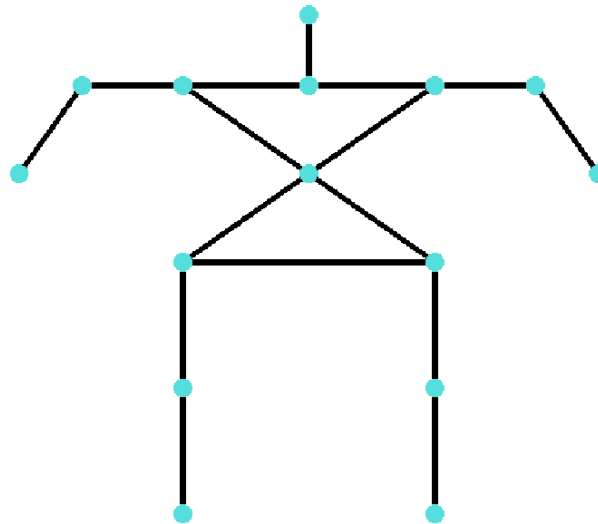


Figure 3.7: Joints in Kinect skeleton

Obstacle detection and avoidance

To begin with, for detecting obstacles, the previously mentioned Kinect skeleton will be used to detect humans. The depth sensor in Kinect will be used to compute the position of objects in 3D space. For avoiding obstacles, a vector is generated from the end effector in the direction opposite to the obstacle to move away from it. Later in the chapters to follow, these two topics will be introduced in details.

Visualization of the system plan is illustrated in Figure 3.8 to familiarize the reader with proposed implementation.

3.3 Control interface

As mentioned in 2, having a user-friendly, user interface is an inevitable option. Murab is a medical project, directed mainly to the field where its first-hand users will be doctors. Mostly, doctors do not have an engineering background, or even a slight coding capability. Nevertheless, it is not recommended nor logical to spend time writing commands for the robot arm to move to a certain position in space when much more economical approaches exist. This could also be useful in testing the system designed and running experiments in a more convenient manner. To achieve this, the ExtremeTM 3D Pro Logitech Joystick3.9 is used to control the manipulator. For cartesian position control, 3 of the 6 axes of the joystick are used to control the end effector in 3D space and a button is used to reset the arm to a pre-set position and orienta-

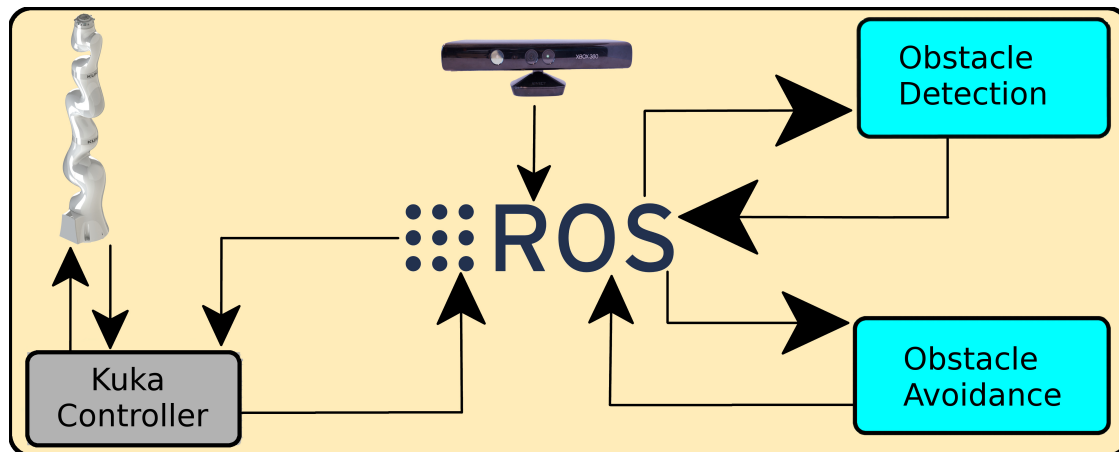


Figure 3.8: Proposed system map

tion. For joint angle control, all available 6 axes of the joystick are used along with 2 buttons to control the 7 joints present in the manipulator. For the complete C++ code written, refer to A.1.



Figure 3.9: ExtremeTM 3D Pro Logitech Joystick

Moreover, the Kinect skeleton illustrated in 3.7 is used to make the arm imitate the user's arm by controlling 4 out of the 7 joints present in the manipulator. For the complete C++ code written, refer to A.2.

4 Environmental awareness

4.1 Obstacle detection methods

In this chapter, I will go through one of the main tasks of this assignment. Obstacle detection is implemented in two approaches. Later on in this report, experiments are carried out to evaluate the reliability of each method.

4.1.1 Body detection

The following detection method is restricted to human detection only. Kinect's OpenNI tracker has the ability to recognize and track a human body. For visualization, a human skeleton is drawn to fit the body tracked and is updated as the body is in motion. The skeleton tracks 15 body joints as seen in Figure 3.7.

kinect_listener is a tf listener file constructed to access the frame transformations of each joint. The frame origin of each joint is relative to the *openni_depthframe* which is basically the kinect frame shown in Figure 4.1. For full code, refer to A.9.

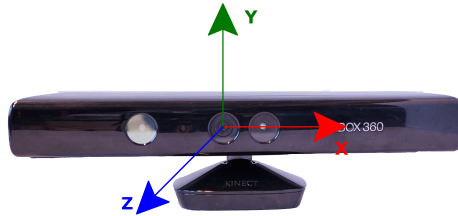


Figure 4.1: Kinect coordinate frame

4.1.2 Depth-pixel detection

The position vector of the manipulator's joints will be used along with a *distance to pixel* algorithm to build a model of the arm using OpenCV drawing functions. Furthermore, OpenCV will use Kinect's depth sensor to get the depth information of all the pixels in the captured image. This is used along with a *pixel to distance* algorithm to get the position vector in 3D space of all the pixels shown by Kinect. Finally, this will be a crucial input to the obstacle avoidance method shown later in this section.

Manipulator model

To build a model for the arm, the pixel information of the joints should be known. Since only the position information for the joints is calculated as illustrated in 5.3.2, a *distance to pixel* approach was used. This approach depends primarily on the depth information of the object in space. Furthermore, the following equations were used to compute the joints' pixel position i and j relative to Kinect.

$$i = ((x/(a \cdot z)) + 320); \quad j = ((-y/(a \cdot z)) + 410) \quad (4.1)$$

x , y and z refer to joint's position on the X , Y and Z axes respectively relative to Kinect coordinate frame. ' a ' is considered a parameter of adjustment that is equal to 0.00173667.

Since Pixel information of the arms' joints are calculated, a model can be built using OpenCV's drawing functions. These functions include

1. Drawing lines.
2. Drawing circles.
3. Drawing rectangles.

As seen in 4.2, the model was generated by drawing lines to match all joints together. Also, a circle was drawn on each joint in a different color to show its position. Furthermore, a dark-filled rectangle was positioned to cover the RGB image of the arm, in a way that the joints are always positioned at its center.

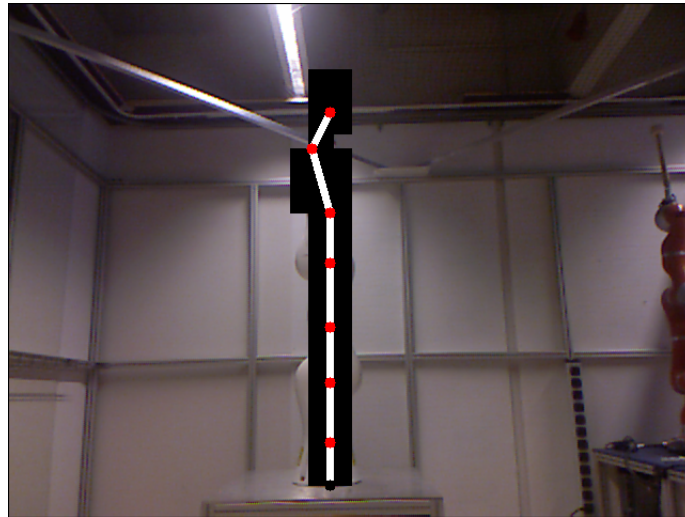


Figure 4.2: Kuka arm OpenCV model

Pixel position vector

The next step is to figure the position information of the objects in the environment monitored by Kinect. This is possible by getting the depth information of all the pixels -640×480 -surveilled. Afterwards, each pixel's i and j can be used along with its respective depth to compute its actual position vector $[x \ y \ z]^T$.

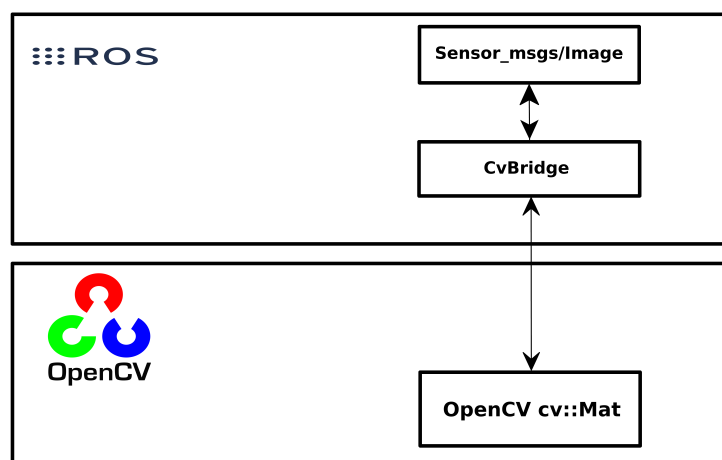


Figure 4.3: Ros to OpenCV via CvBridge

To get the depth information from Kinect's depth sensor, **cv_bridge** must be used. This is a package to communicate between ROS and OpenCV. Ros publishes depth information of type known as *sensor_msgs/Image*, but unfortunately it is unreadable information. **cv_bridge** makes a copy of this message and passes it on to OpenCV in **cv::Mat** format to be decoded. Afterwards, it publishes it again to ROS as a matrix of size 640×480 with the depth information needed. This process is further illustrated in Figure 4.3. Since there now exists a matrix with depth information for all pixels, next step is to get the position vector of each pixel in 3D space.

This approach is similar to that used in 4.1, however i and j are now inputs, and x and y are required. Accordingly, they should be the subjects of the equation. This is illustrated as follows.

$$x = 2360 - k; \quad y = (i - 320) \cdot a \cdot k; \quad z = -(j - 410) \cdot a \cdot k \quad (4.2)$$

k is the depth measurement computed from Kinect's depth sensor. Now that we have each pixel's position vector in 3D space, obstacle avoidance can be illustrated as shown in the final part of this section.

5 Obstacle avoidance algorithm

5.1 Notation definitions

Table 5.1: Table of Notation My Research

Ψ^K	\triangleq	Kinect coordinate system
Ψ^B	\triangleq	Kuka base coordinate system
\mathbf{H}_i^j	\triangleq	Homogeneous matrix from frame i to frame j .
R_i^j	\triangleq	Rotation matrix from frame i to frame j .
\mathbf{d}_i^j	\triangleq	Translation matrix from frame i to frame j .
T_0^n	\triangleq	Transformation matrix from frame manipulator's base frame to frame n .
A_i	\triangleq	Homogeneous matrix from frame $i - 1$ to frame i .
T_i^j	\triangleq	Homogeneous transformation matrix from frame j to frame i .

5.2 Kinect to Kuka frame transformation

Since all the coordinates of the skeleton joints are relative to the Kinect Coordinate System (Ψ^K), a Homogeneous Matrix is calculated and applied to obtain the position vectors of the skeleton joints with respect to the Manipulator Base Coordinate System (Ψ^B). The Homogeneous Matrix combines both the rotation and translation operations in one 4×4 matrix. The equation for the Homogeneous Matrix, \mathbf{H} , is

$$\mathbf{H}_i^j = \begin{bmatrix} R_i^j & \mathbf{d}_i^j \\ \mathbf{0} & 1 \end{bmatrix} \quad (5.1)$$

In the following Figure 5.1, rotation from Kinect coordinate frame to Kuka coordinate frame is shown. It starts with rotating the frame around the X-axis with an angle of -90° and then rotating around the Z-axis with an angle of 90° .

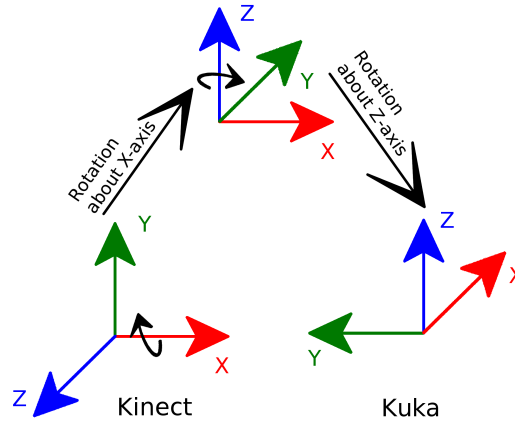


Figure 5.1: Rotation from Kinect to Kuka coordinate frames

In the following figure 5.2, translation from Kinect coordinate frame to Kuka coordinate frame is shown.

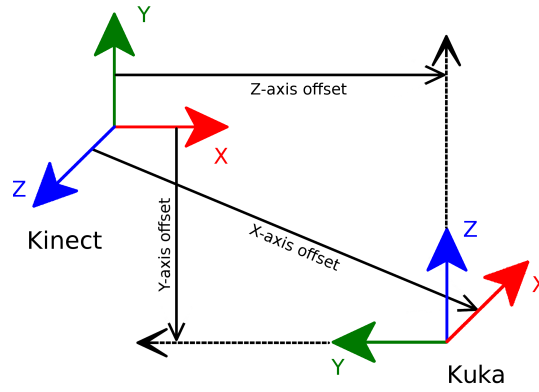


Figure 5.2: Translation from Kinect to Kuka coordinate frames

$$\mathbf{d}_i^j = \begin{bmatrix} \mathbf{d}_{i_x}^j & \mathbf{d}_{i_y}^j & \mathbf{d}_{i_z}^j \end{bmatrix}^T \quad (5.2)$$

where R_i^j is the 3×3 rotation matrix from coordinate system i to coordinate system j , and \mathbf{d}_i^j is the 3×1 translation matrix between them. Rotation, Transformation and resulting Homogeneous matrices are illustrated for our system in the following equation.

$$R_K^B = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{d}_K^B = \begin{bmatrix} 2.34 \\ 0 \\ 0.8 \end{bmatrix} \quad \Rightarrow \quad H_K^B = \begin{bmatrix} 0 & 0 & -1 & 2.34 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.3)$$

Position vectors are modified to comply with the dimensions of the Homogeneous Matrix.

$$P_K = \begin{bmatrix} x_K & y_K & z_K & 1 \end{bmatrix}^T \quad P_B = \begin{bmatrix} x_B & y_B & z_B & 1 \end{bmatrix}^T \quad (5.4)$$

If the position vector of a point \mathbf{P}_K is known in coordinate system (Ψ^K) , it can be represented in coordinate system (Ψ^B) as \mathbf{P}_B using

$$P_B = \mathbf{H}_B^K \cdot P_K \quad (5.5)$$

where \mathbf{H}_B^K is the Homogeneous Matrix from coordinate frame (Ψ^K) to coordinate frame (Ψ^B) .

This is illustrated in our system as follows

$$\begin{bmatrix} x_B \\ y_B \\ z_B \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 2.34 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.8 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_K \\ y_K \\ z_K \\ 1 \end{bmatrix} \quad (5.6)$$

Using WxMaxima, the computed matrix was simulated and tested, giving positive results. Furthermore, for the C++ function used to transform the joints' position vectors from one frame to the other, refer to A.5.

5.3 Forward kinematics - Kuka arm

5.3.1 Denavit-Hartenberg convention

The next topic tackled in this approach is the Forward Kinematics, which is basically the use of kinematic equations with joint angles being the equations' input, to calculate the position vector of end-effector. The Denavit-Hartenberg was the method used for Forward Kinematics, where coordinate axes for each joint should be chosen that the following 2 features are present:

1. Axis x_i should be perpendicular to axis z_{i-1} .
2. Axis x_i should intersect axis z_{i-1} .

After choosing the axes, it is necessary to measure a , α , d and θ . a is the distance between axes z_i and z_{i-1} , α is the angle between axes z_i and z_{i-1} in plane normal to x_i , d is the distance between the origin O_{i-1} and the intersection of x_i with z_{i-1} and finally θ is the angle between x_i and x_{i-1} in a plane normal to z_{i-1} . The following table shows the Denavit-Hartenberg parameters measured for the Kuka robotic arm for each link i :

i	a	α	d	θ_i
1	0	$\pi/2$	360	θ_1
2	0	$-\pi/2$	0	θ_2
3	0	$-\pi/2$	420	θ_3
4	0	$\pi/2$	0	θ_4
5	0	$\pi/2$	400	θ_5
6	0	$-\pi/2$	0	θ_6
7	0	0	126	θ_7

Table 5.2: Denavit-Hartenberg Parameters

Using the parameters above in the equation below, the homogeneous transformation for each joint can be calculated.

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.7)$$

Finally, to calculate the position of the end-effector, transformation matrices for the n links should be multiplied all together.

$$T_0^n = A_1 \cdots A_n \Rightarrow T_0^7 = A_1 A_2 A_3 A_4 A_5 A_6 A_7 = \begin{bmatrix} R_{3 \times 3} & d_{3 \times 1} \\ 0 & 1 \end{bmatrix} \quad (5.8)$$

where $d_{3 \times 1}$ is the position matrix of the end-effector relative to the base of the manipulator.

$$d_{3 \times 1} = \begin{bmatrix} d_x & d_y & d_z \end{bmatrix}^T \quad (5.9)$$

For a more detailed mathematical visualization of the matrices computed for the manipulator, refer to A.3

5.3.2 Translation and Rotation method

As mentioned in 5.1, Homogeneous matrix \mathbf{H}_i^j consists of a Rotation matrix R_i^j and a Translation matrix \mathbf{d}_i^j . On the contrary, Translation and Rotation matrices are calculated separately as a transformation from each joint coordinate frame on the manipulator to the next. Translation matrices are illustrated as follows.

$$\mathbf{d}_{i+1}^i = \begin{bmatrix} 1 & 0 & 0 & x_{i+1}^i \\ 0 & 1 & 0 & y_{i+1}^i \\ 0 & 0 & 1 & z_{i+1}^i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.10)$$

x_{i+1}^i , y_{i+1}^i and z_{i+1}^i are the offset values in the X , Y and Z directions respectively.

Furthermore, after translation is calculated from coordinate frame i to coordinate frame $i + 1$, frame $i + 1$ is to be rotated to match the next joint coordinate frame, $i + 2$. Encountered rotations were limited to only either **Yaw** or **Pitch** rotations. **Yaw** rotations, R_{i+2}^{i+1} represent rotations taking place about the $Z - axis$ by an angle ψ from coordinate frame $i + 1$ to coordinate frame $i + 2$ as illustrated below.

$$R_{i+2}^{i+1} = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.11)$$

On the other hand, **Pitch** rotations, R_{i+2}^{i+1} represent rotations taking place about the $Y - axis$ by an angle θ from coordinate frame $i + 1$ to coordinate frame $i + 2$. This is also illustrated below.

$$R_{i+2}^{i+1} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (5.12)$$

For a 7 link manipulator, 7 Translation matrices and 8 Rotation matrices are multiplied together to get the position vector of the end effector relative to the base of the robot arm. For the

robot arm's base being coordinate frame 0 and the end effector being coordinate frame 15, the homogeneous transformation, \mathbf{T} , is calculated as follows.

$$\mathbf{T}_{15}^0 = \mathbf{d}_1^0 R_2^1 \mathbf{d}_3^2 R_4^3 \cdots \mathbf{d}_{13}^{12} R_{14}^{13} \mathbf{d}_{15}^{14} \Rightarrow \begin{bmatrix} n_x & s_x & a_x & \mathbf{d}_x \\ n_y & s_y & a_y & \mathbf{d}_y \\ n_z & s_z & a_z & \mathbf{d}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n & s & a & \mathbf{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.13)$$

In the equation 5.13, $n = [n_x \ n_y \ n_z]^T$, $s = [s_x \ s_y \ s_z]^T$ and $a = [a_x \ a_y \ a_z]^T$ refers to the direction of end effector's X , Y and Z axes respectively as seen relative to the arm's base frame. More importantly, the position vector of the end effector relative to the arm's base is illustrated in the following equation.

$$\mathbf{d} = [\mathbf{d}_x \ \mathbf{d}_y \ \mathbf{d}_z]^T \quad (5.14)$$

Furthermore, the previous Translation and Rotation matrices can be used to calculate the position vector of all joints of the manipulator. For an unambiguous illustration, equation 5.13 will be modified to be only the result of multiplying 8 matrices instead of 15. This will be satisfied by calculating \mathbf{T}_{15}^0 as follows.

$$\mathbf{T}_{15}^0 = \mathbf{T}_2^0 \mathbf{T}_4^2 \mathbf{T}_6^4 \mathbf{T}_8^6 \mathbf{T}_{10}^8 \mathbf{T}_{12}^{10} \mathbf{T}_{14}^{12} \mathbf{T}_{15}^{14}, \quad (5.15)$$

where \mathbf{T}_j^i is calculated as follows.

$$\mathbf{T}_{i+2}^i = \mathbf{d}_{i+1}^i R_{i+2}^{i+1}; \quad \mathbf{T}_{i+1}^i = \mathbf{d}_{i+1}^i \quad (5.16)$$

For a more detailed mathematical visualization of the matrices computed for the manipulator, refer to A.4

For calculating the position vector of joint i , T_{i*2}^0 should be computed as shown in 5.15. **i.e.** joint 3 position vector is equal to \mathbf{d} from 5.14 extracted from $T_{3*2}^0 = T_6^0$, where $T_6^0 = \mathbf{d}_1^0 R_2^1 \mathbf{d}_3^2 R_4^3 \mathbf{d}_5^4 R_6^5$.

Finally, using the mathematical equations computed in this subsection, and only joint angles as inputs, not only the end effector's but all the joints' position vectors can be calculated. This will be quite useful for the approach and visualization shown in the rest of this section.

5.4 Inverse kinematics - Kuka arm

Position cartesian commands passed to the system can only be given to the end effector. This means that all the joints of the manipulator can not be position controlled in 3D space but only controlled through passing joint angles. Figure 5.3 further illustrates the problem.

As seen in the above figure, the black arrows show how cartesian commands can only be passed to the end effector. On the other hand, the red arrows show how only joint angle commands can be passed to each joint.

The method used to overcome this issue is computing the Inverse Kinematics. Inverse Kinematics is the opposite of Forwards Kinematics. It consists of a number of kinematic equations

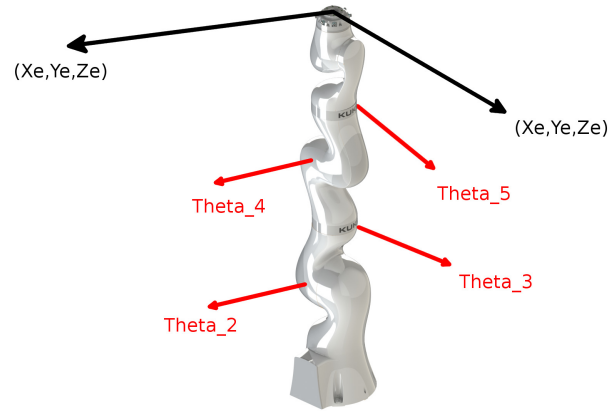


Figure 5.3: Manipulator control approaches

where the input to these equations is the Position vector $[x_{ef} \ y_{ef} \ z_{ef}]^T$ of the end effector in 3D space relative to a certain coordinate frame. The output of these equations would be the joint angles needed for the end effector to be at this position. Inverse Kinematics in general is more difficult to compute than the Forward Kinematics, specially when the system examined is a 7 DOF kinematically redundant manipulator. Kinematic redundancy of the manipulator unfortunately results in inverse kinematics not working all the time. Nevertheless, sometimes non-linearity of the equations requires high computation time which would need an extremely powerful processor to calculate the joint angles. Luckily, several approaches to compensate for the inverse kinematics' complications for redundant manipulators have been tested previously. As discussed in (Buss, 2004), some of these method are as follows.

1. Jacobian Transpose Method
2. Pseudoinverse Method
3. Damped Least Squares Method

Methods like Pseudoinverse and Damped Least Squares may be more accurate than the Jacobian Transpose method. On the other hand, Jacobian Transpose has way faster computation speed. Although this would have been suitable for our system, but Jacobian Transpose method not only results in unexpected joint angles when the input is a point outside its workspace, but also fails most of the time at singularities. As mentioned in (Mark W. Spong, 1989), near singularities there will be either no or infinite solutions for the inverse kinematics problem. Singularities are basically points within the manipulator's workspace where two joints line up turning them redundant.

Since our system is only interested in moving away from the obstacle, and not necessarily going to a certain position in space, high reliability of the joint angles computed is not needed. The approach suggested reduces the current 7-Link manipulator to a 4-Link model instead. This will result in an easier to compute inverse kinematics, although it will result in having some restrictions in the manipulator's flexibility in comparison to the regular 7-Link manipulator.

Inverse Kinematics

Three equations are used to calculate 3 joint angles needed to for end effector to reach a certain position in space. These equations are illustrated in 5.17, 5.18 and 5.19. Furthermore, to calculate the joint angles needed for a certain joint to move in space to a certain position, an even

reduced inverse kinematics can be calculated where the model of the system can be consisting of 3 links instead of 4.

$$\theta_1 = \text{atan2}(P_x, P_y) - \frac{\pi}{2} \quad (5.17)$$

$$\theta_3 = \frac{\pi}{2} - \text{atan2}(D, \sqrt{1 - D^2}) \quad (5.18)$$

$$\theta_2 = - \left[\text{atan2}(\sqrt{P_x^2 + P_y^2}, P_z) - \text{atan2}(a_2 + a_3 \cos \theta_3, a_3 \sin \theta_3) \right] - \frac{\pi}{2} \quad (5.19)$$

P_x , P_y , and P_z refer to the end effector's position coordinates and a_2 and a_3 refer to the manipulator's links' length. They are all used to compute D which is calculated as shown in 5.20.

$$D = \frac{P_x^2 + P_y^2 + P_z^2 - a_2^2 - a_3^2}{2a_2a_3} \quad (5.20)$$

An illustration of the variables used in the previous description is shown in Figures 5.4 and 5.5.

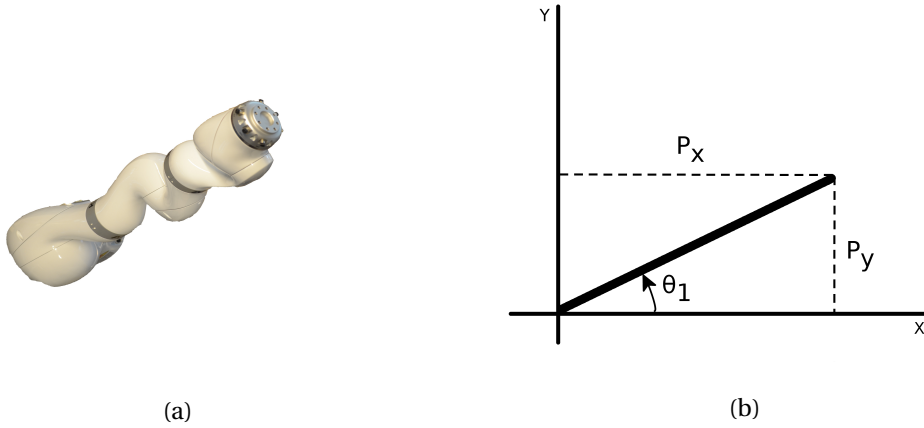


Figure 5.4: Top view illustration of model used for Inverse Kinematics. (a) Manipulator's orientation. (b) Respective model for orientation in (a).

5.5 Avoidance algorithm

It is assumed that there exists an invisible circular area around the arm's end-effector. This area, which will be called the safety area, is of radius 500 mm and the optimum goal will be for the end-effector to drift away from any obstacle interfering with its safety area. 500 mm is a random choice of radius and was only chosen for testing.

To begin with, as illustrated in A.6, method `getDistance` calculates the distance between 2 points in space. It will be used to get the geometric distance between joints in the human skeleton and the manipulator's end-effector.

This is done using a simple geometric concept where distance d between 2 points in space can be illustrated as

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (5.21)$$

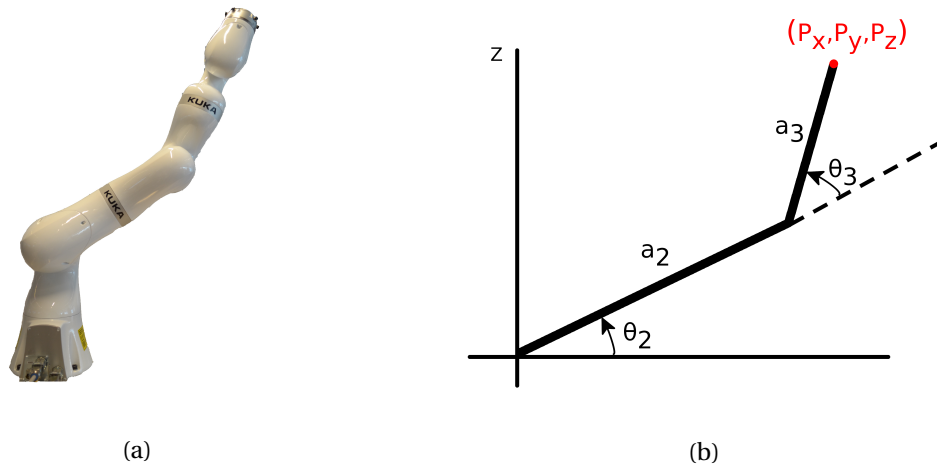


Figure 5.5: Side view illustration of model used for Inverse Kinematics. (a) Manipulator's orientation. (b) Respective model for orientation in (a).

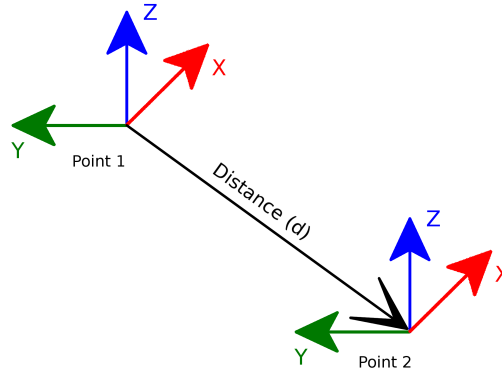


Figure 5.6: Distance between Point 1 and Point 2

where $[x_1 \ y_1 \ z_1]^T$ is the position vector of point 1 and $[x_2 \ y_2 \ z_2]^T$ is the position vector of point 2 as shown in Figure 5.6.

The robot arm will be controlled in this approach using Cartesian position commands. By passing Position and Orientation coordinates to manipulator's end-effector, it can smoothly move in space.

1 Obstacle

Regarding controlling the safety of the manipulator, if an object's distance from the end-effector is less than 500 mm, an equal vector is to be generated in the opposite direction to the obstacle. This is illustrated thoroughly in Figure 5.7.

As seen in the preceding figure, the blue arrow shows the vector generated from the obstacle to the arm's end effector. The green arrow, on the other hand, reflects the vector generated in the opposite direction to move the end effector towards.

Moreover, to choose exactly which point on the vector the end effector should move to, a factor is to be computed according to how far the end effector is from the object. This factor is multiplied by the unit vector generated and the result is the position coordinates of the point in 3D space. Following the A.6 function, the factor approach is included in A.7.

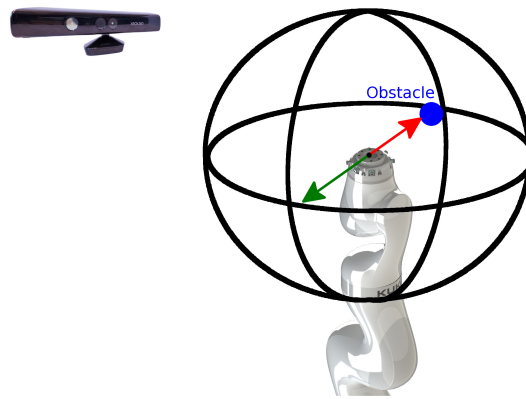


Figure 5.7: Illustration of 1 obstacle avoidance method

For clarification, the following Flowchart 5.8 explains how the factor is chosen.

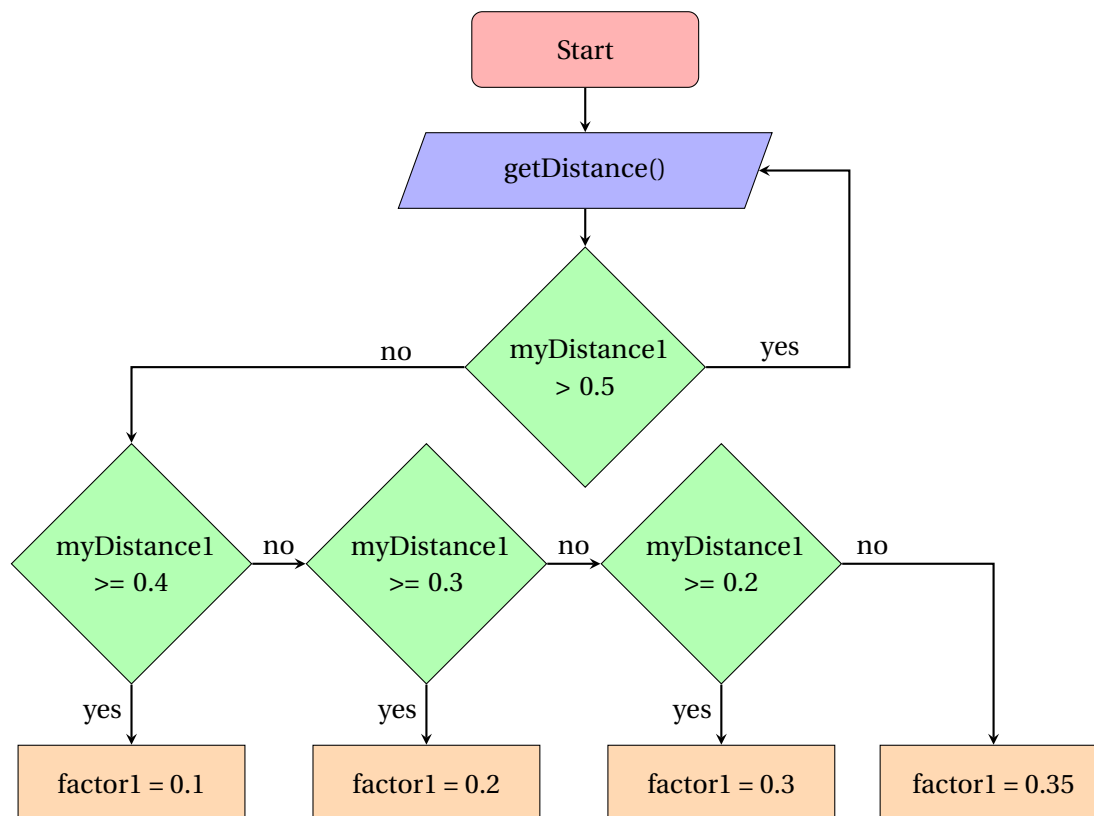


Figure 5.8: Choosing factor for cartesian command

2 Obstacles

For a more powerful safe environment, another algorithm was computed to dodge 2 obstacles if they are both located in the safety area at the same time. This is illustrated graphically in the following Figure 5.9.

As seen in Figure 5.9, the green vector generated is the resultant direction of the two red vectors generated due to the two obstacles.

Following calculating the distance between the end effector and the obstacles, the code in A.8

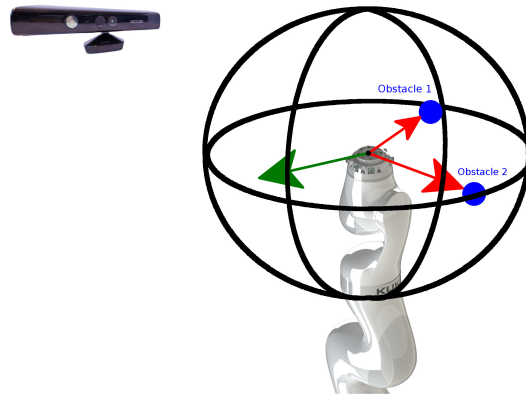


Figure 5.9: Illustration of 2 obstacles avoidance method

illustrates the algorithm used to compute the position vector that should be passed to the end effector.

Furthermore, the factor that is multiplied by the unit vector generated is computed in a slightly different manner. The idea behind this factor is to choose a certain point in the direction of the vector generated according to how near the obstacle is from the end effector. A snippet of the code where the factor is computed, followed by a graph 5.10 showing how the relation between the factor and the distance of the object are both illustrated as follows.

```

if (myDistance1 <= maxDistance1)
    factor1 = ((maxDistance1 - myDistance1) / myDistance1);
else
    factor1 = 0;

```

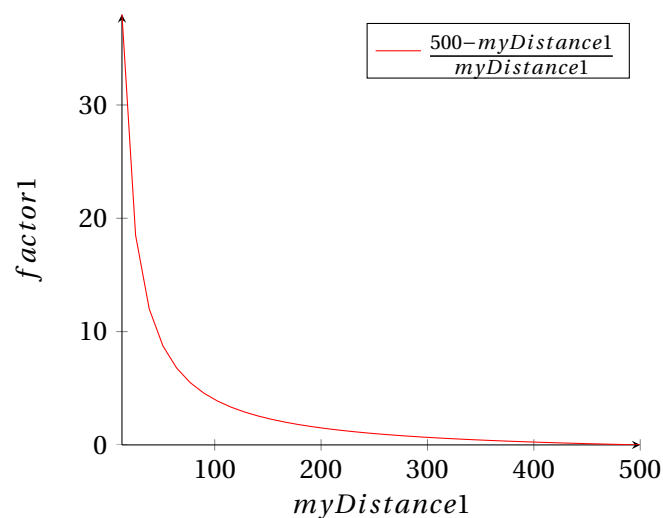


Figure 5.10: A plot of the factor computed against the distance calculated

Recap

The subsequent Flowchart 5.11 recaps the steps followed in the aforementioned approach.

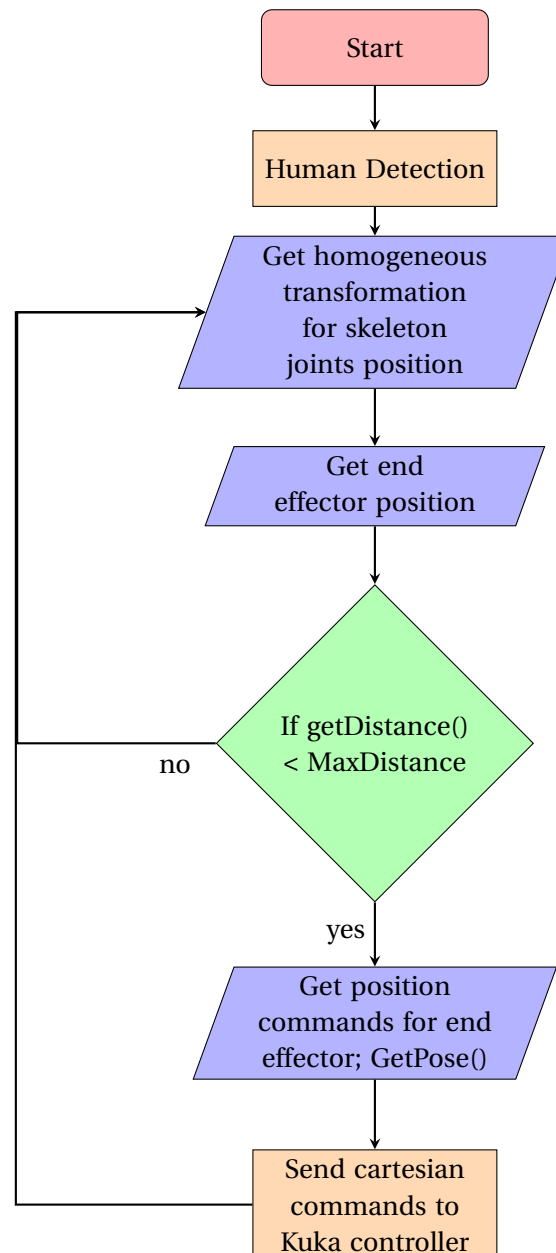


Figure 5.11: Body Detection

The following Flowchart 5.12 shown gives a summary of the steps the second approach goes through.

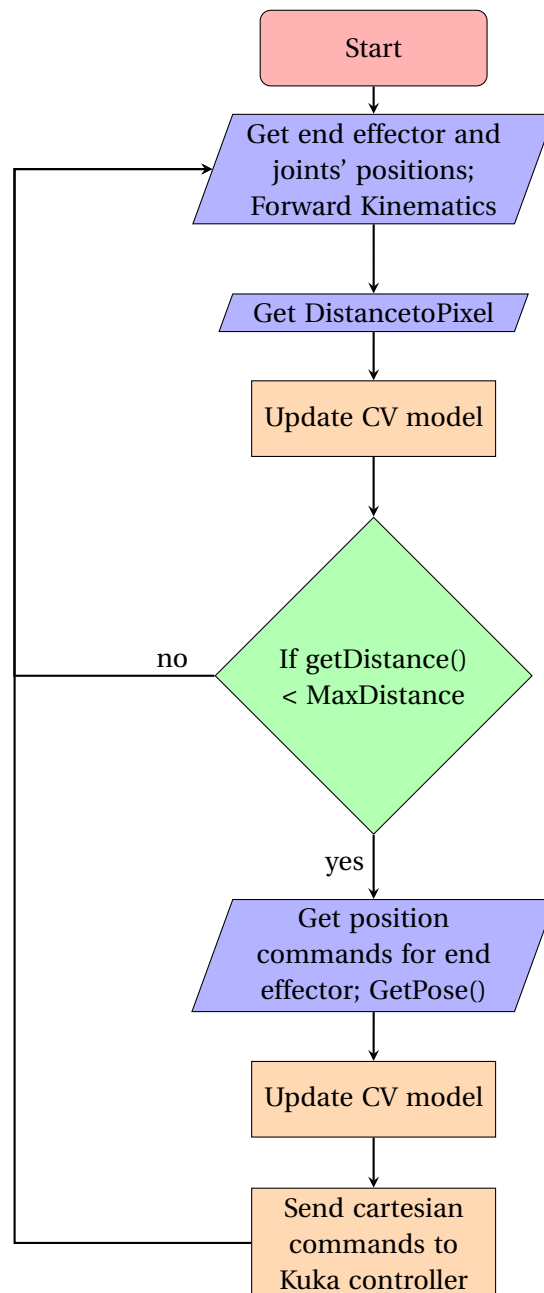


Figure 5.12: Depth-pixel detection

6 Experiments and results

In the following chapter, the proposed implementations are tested, evaluated and analysed. Each method experimented is mentioned and its results are discussed afterwards.

6.1 Obstacle detection and avoidance

To begin with, all the experiments took place in the RaM lab in the University of Twente. The manipulator is placed in a 3.6×3.6 metre cage, with a height of 2.3 metre. The obstacle avoidance approach is tested firstly along with the body detection method of awareness. Kinect is placed at 2.34 metres away from the arm's base as shown in Figure 6.1.



Figure 6.1: Experiment setup

Two individuals take part in this experiment, where one, who is to be called "A" can control the system and check the results and the other, who is to be called (b) is actual part of the avoidance implementation. After (b) is detected by Kinect, an arm motion is executed towards the end effector from North-East to South-West, as seen by Kinect. The motion also included shifting away from Kinect. The full motion is illustrated in the following Figure.6.6.

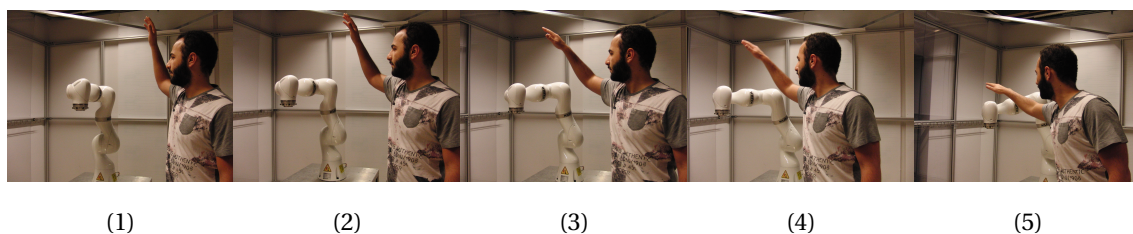
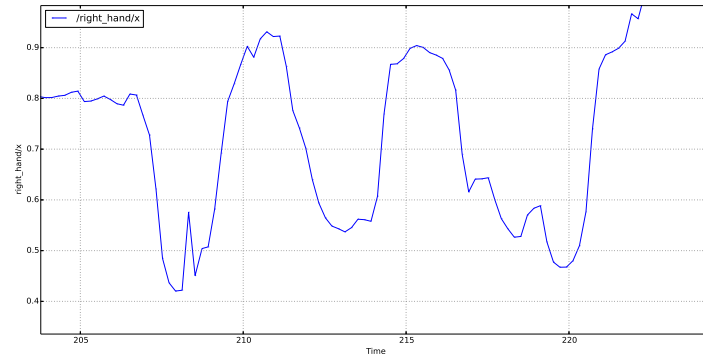
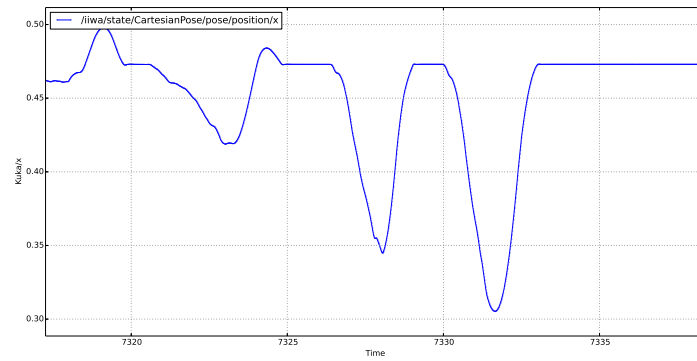


Figure 6.2: Body detection and obstacle avoidance in frames

Furthermore the positions of the end effector and the right hand joint of (b) are plotted. Unfortunately, due to a software complication with ROS, the Time axis of the end effector's position in the experiments was not in synchronization with the Time axis of the right hand position. Although this is a major deficiency in the result's verification, response time of the obstacle avoidance method was not the main focus here.

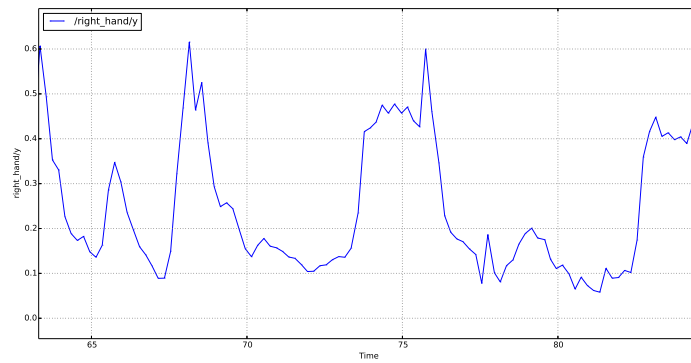


(a)

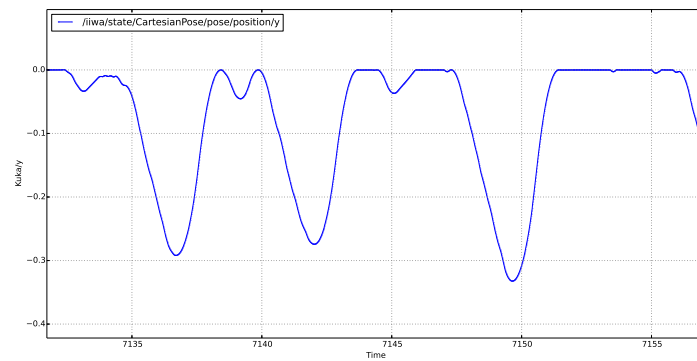


(b)

Figure 6.3: X-axis plots. (a)Right hand position in x-direction. (b)End effector position in x-direction



(a)



(b)

Figure 6.4: Y-axis plots. (a)Right hand position in y-direction. (b)End effector position in y-direction

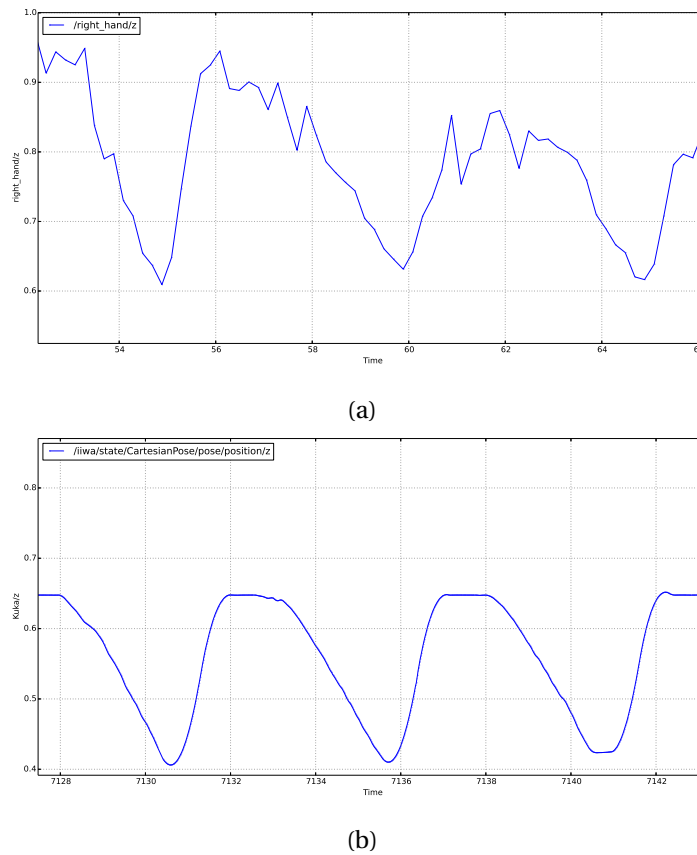


Figure 6.5: Z-axis plots. (a) Right hand position in z-direction. (b) End effector position in z-direction

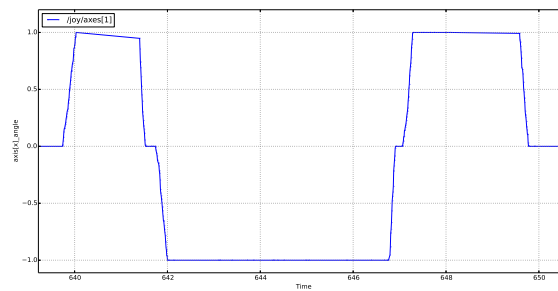
It is quite impossible to identify the accuracy of the system developed due to the problem with Time synchronization. Furthermore, it has been observed that the skeleton detected can sometimes be lost when not all the human body is visible to Kinect. On the other hand, results show acceptable precision as the experiment was repeated more than 10 times and 90% of the plots generated were similar. Moreover, three different individuals, with heights varying from 160, 180 and 190 metres have played the role of individual (b) in the experiment and the system gave results similar to the plots above. This shows that the system is reliable and would give the same results with physically different human beings.

Secondly, the obstacle avoidance approach is tested along with the depth-pixel detection method. Results were not very promising as the obstacle position information published were not accurate. This happens mainly due to the non-aligned RGB and depth images giving false position vectors of the pixels captured. Another issue showed up as well, which was that the end effector was detecting the manipulator's body as an obstacle. This resulted in passing cartesian space commands that forces the arm to inevitably lock, because it is like that the end effector is trying to move away from the arm. On the contrary, when this approach was tested in plain space with the actual arm not present, the built OpenCV model showed quite precise results in avoiding obstacles. Although the RGB and the depth images need to be aligned for high accuracy, the approach itself shows expected results in terms of detecting obstacles.

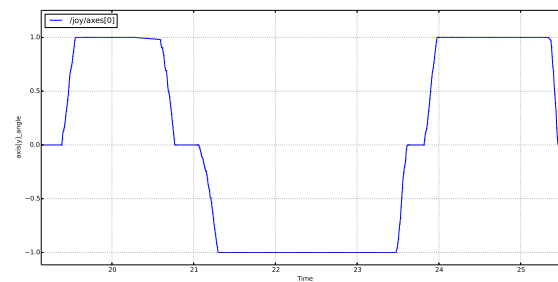
Unfortunately as a result to this method not working, the inverse kinematics could not be practically tested. Nevertheless, the 3 equations computed to calculate the 3 joint angles were tested with the arm's controller as a reference. The 3 angles are computed using the end effector cartesian position which is accessible at all times. Afterwards, the angles are checked with their respective angles on the manipulator which are measured using accurate angle encoders. The inverse kinematics returned similar results to the ones obtained from the arm's controller.

6.2 Control interface

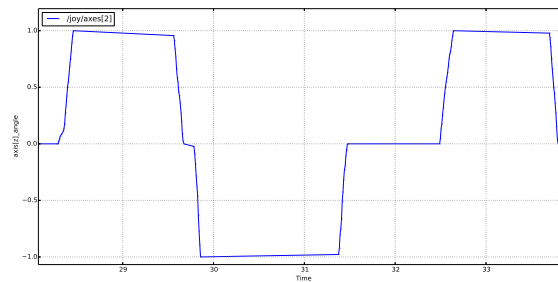
Using the joystick was an adequate approach and showed utter practicality when in use. Passing joint angles through the joystick returned accurate results and was seen to be a reliable means of control when tested. It is unfortunately only used in limited environments where joint angle control is necessary. On the other hand, cartesian space control using the joystick did not return perfect results at all times. The arm's controller failed to compute the inverse kinematics due to a high unit step chosen at the beginning which was further improved afterwards by limiting the unit step to a certain minimum. This unfortunately caused some limitations in the maximum velocity reached by the manipulator. Controlling the end effector's cartesian position using the Logitech joystick was tested and results were plotted as follows. The end effector was moved first along the x-axis then along the y-axis then along the z-axis. This was followed by random motion in space then the reset button was pushed for the arm to go to a pre-set position.



(a)



(b)



(c)

Figure 6.6: Joystick axes angles. (a)x-axis (b)y-axis (c)z-axis.

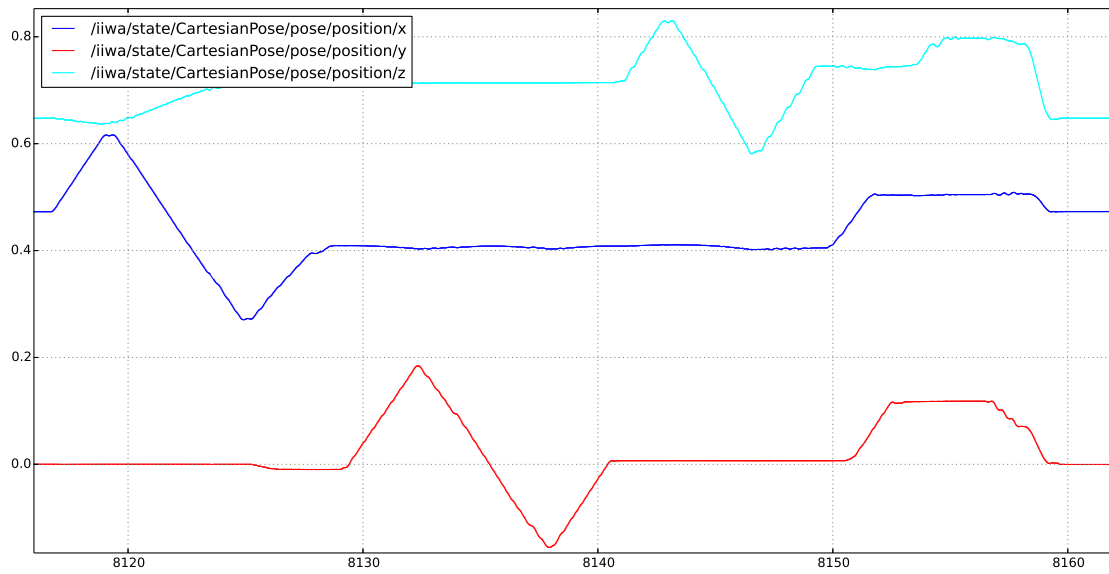
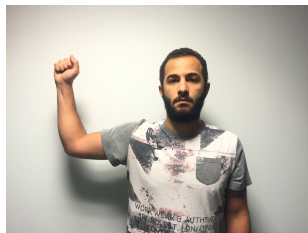


Figure 6.7: End effector X, Y and Z position.

Arm imitation method of controlling the arm showed a number of imperfections. To begin with, due to technical defects in Kinect, depth information showed unexpected fluctuations which caused the joints' positions to fluctuate wrongly. Using arm imitation, only 4 joints of the manipulator were controlled which restricted the use of all 7 links. Finally, the imitate control interface was developed and 4 joints of the manipulator are controlled by copying the motion of a human arm. The joints controlled are illustrated in the Figures below.



(a)

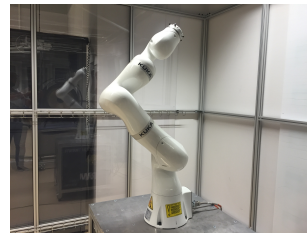


(b)

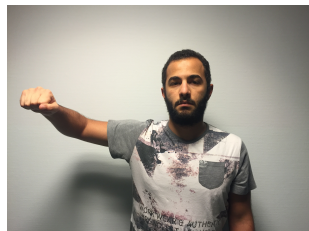
Figure 6.8: Robot arm copying human's arm.



(a)



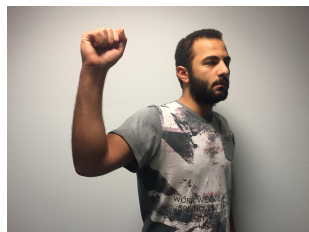
(b)



(c)



(d)



(e)



(f)



(g)



(h)

Figure 6.9: Human arm controlling 4 of the manipulator's joints

7 Conclusion and recommendations

7.1 Conclusion

An intensive literature review on safety guidelines needed for medical robotics was done. This literature survey was used to ensure that having an obstacle avoidance system was quite crucial for a medical robot to exist safely. A system was designed for detecting and avoiding obstacles appearing in the arm's environment. This included the use of Xbox 360 Kinect sensor along with Kuka robot arm. A description on how to operate all the system's proposed implementations was composed. The first obstacle detection method works as expected but is limited to detecting only human beings. The second obstacle detection method did not give the most promising results due to calibration complications with Kinect. Obstacle avoidance method gave the expected precise results. It relies mainly on calculating the geometric distance between 2 points in 3D space and then generating a vector in the opposite direction to the obstacle point and sending commands to the end effector to move this way. An inverse kinematics approach was attempted to not only have the end effector avoid obstacles but also the arm's joints. Although it could not be tested practically as the second obstacle detection method was not working, it gave the expected results when tested separately. With a number of improvements in several areas, it is believed that the performance of the designed system could be ameliorated.

7.2 Recommendations

To begin with, I believe using 2-4 depth sensors could highly support the validation of the designed approach. Further software research on the topic real-time response is highly advisable. A method for avoiding any number of obstacles could be tested, by adding all the vectors from the end effector to the obstacles and then generating a vector in the opposite direction. This was not implemented due to lack of time. Extensive tests on the depth sensor in use should take place to assure the most accurate facilities are in use. Finally, the literature survey done should be thoroughly looked into as it is basically made up of future work for safety measures that should be taken in this kind of projects.

A Appendix 1

A.1 control_iiwa_1.cpp

```

#include "ros/ros.h"
#include "iiwa_msgs/JointPosition.h"
#include "geometry_msgs/PoseStamped.h"
#include <std_msgs/Float64.h>
#include <sensor_msgs/Joy.h>

using namespace std;

iiwa_msgs::JointPosition current_joint_position, command_joint_position;
geometry_msgs::PoseStamped current_cartesian_position, command_cartesian_position;
std::string joint_position_topic, cartesian_position_topic,
    ↪ command_cartesian_position_topic, command_joint_position_topic;
sensor_msgs::Joy joy; double x, y, z, j_1, j_2, j_3, j_4, j_5, j_6, j_7p, j_7n,
    ↪ reset; int control = 0;

double ros_rate = 30.0; bool isRobotConnected = true;

void jointPositionCallback(const iiwa_msgs::JointPosition& jp)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    current_joint_position = jp;
}

void cartesianPositionCallback(const geometry_msgs::PoseStamped& ps)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    current_cartesian_position = ps;
}

void joystickCallback(const sensor_msgs::Joy& js)
{
    joy = js;
    j_1 = x = joy.axes[1];
    j_2 = y = joy.axes[0];
    j_3 = z = joy.axes[2];
    j_4 = joy.axes[4];
    j_5 = joy.axes[5];
    j_6 = joy.axes[3];
    j_7p = joy.buttons[1];
    j_7n = joy.buttons[0];
    reset = joy.buttons[11];
}

int main (int argc, char** argv) {

    ros::init(argc, argv, "control_iiwa_1");
    ros::NodeHandle nh("~");
    ros::AsyncSpinner spinner(1);
    spinner.start();
    nh.param("joint_position_topic", joint_position_topic, std::string("/iiwa/
    ↪ state/JointPosition"));

```

```

nh.param("cartesian_position_topic", cartesian_position_topic, std::string(
    ↪ "/iiwa/state/CartesianPose"));
nh.param("command_cartesian_position_topic",
    ↪ command_cartesian_position_topic, std::string("/iiwa/command/
    ↪ CartesianPose"));
nh.param("command_joint_position_topic", command_joint_position_topic, std
    ↪ ::string("/iiwa/command/JointPosition"));
nh.param("ros_rate", ros_rate); // 0.1 Hz = 10 seconds
ros::Rate* loop_rate_ = new ros::Rate(ros_rate);

// Subscribers and publishers
ros::Subscriber sub_joint_position = nh.subscribe(joint_position_topic, 1,
    ↪ jointPositionCallback);
ros::Subscriber sub_cartesian_position = nh.subscribe(
    ↪ cartesian_position_topic, 1, cartesianPositionCallback);
ros::Subscriber sub_joystick = nh.subscribe("/joy", 1, joystickCallback);

ros::Publisher pub_cartesian_command = nh.advertise<geometry_msgs::
    ↪ PoseStamped>(command_cartesian_position_topic, 1);
ros::Publisher pub_joint_command = nh.advertise<iiwa_msgs::JointPosition>(
    ↪ command_joint_position_topic, 1);

while (ros::ok()) {
    if (isRobotConnected) {

if(control == 0){
cout << "PLEASE_ENTER_CONTROL_MODE\n.\n.\n1_for_Cartesian_control.\n2_for
    ↪ _Joint_control.\n\n";
cin >> control;
}
command_joint_position = current_joint_position; command_cartesian_position
    ↪ = current_cartesian_position;

if(control == 1){
command_cartesian_position = current_cartesian_position;
command_cartesian_position.pose.position.x += x / 50;
command_cartesian_position.pose.position.y += y / 50;
command_cartesian_position.pose.position.z += z / 50;
ROS_INFO("New_Cartesian_Position_is:_[%3.3f,%3.3f,%3.3f,%3.3f,%3.3f,%
    ↪ %3.3f,%3.3f]",
        command_cartesian_position.pose.position.x,
        command_cartesian_position.pose.position.y,
        command_cartesian_position.pose.position.z,
        command_cartesian_position.pose.orientation.x,
        command_cartesian_position.pose.orientation.y,
        command_cartesian_position.pose.orientation.z,
        command_cartesian_position.pose.orientation.w);
pub_cartesian_command.publish(command_cartesian_position);

if(reset > 0){
command_joint_position.position.a1 = 0;
command_joint_position.position.a2 = 0.174533; //10 degrees
command_joint_position.position.a3 = 0;
command_joint_position.position.a4 = -1.39626; //-80 degrees
command_joint_position.position.a5 = 0;
command_joint_position.position.a6 = 1.5708; //90 degrees
command_joint_position.position.a7 = 0;
pub_joint_command.publish(command_joint_position);
}
}
else if(control == 2){
command_joint_position = current_joint_position;

```

```

command_joint_position.position.a1 += j_1 / 10;
command_joint_position.position.a2 += j_2 / 10;
command_joint_position.position.a3 += j_3 / 10;
command_joint_position.position.a4 += j_4 / 10;
command_joint_position.position.a5 += j_5 / 10;
command_joint_position.position.a6 += j_6 / 10;
command_joint_position.position.a7 += j_7p / 10;
command_joint_position.position.a7 -= j_7n / 10;
ROS_INFO("Current_Joint_Position_is: [%3.3f, %3.3f, %3.3f, %3.3f, %3.3f, %3.3f, %3.3f]",
    ↪ %3.3f, %3.3f]",
        command_joint_position.position.a1,
        command_joint_position.position.a2,
        command_joint_position.position.a3,
        command_joint_position.position.a4,
        command_joint_position.position.a5,
        command_joint_position.position.a6,
        command_joint_position.position.a7);
pub_joint_command.publish(command_joint_position);
}
else {
cout << ".\n.\n.\n.\n";
ROS_WARN("YOU_HAVE_ENTERED_A_WRONG_RESPONSE.");
cout << ".\n.\n.\n.\n";
control = 0;
}

        loop_rate_ ->sleep();
    }
    else {
        ROS_ERROR("Robot_is_not_connected...");
        ros::Duration(1.0).sleep(); // 5 seconds
    }
}

std::cerr<<"\n.\n.\n.\n.\n.\n.\n.\nExiting_control_iiwa_1\nPlease_wait_\n.\n
    ↪ .\n.\n.\n.\n."<<std::endl;
spinner.stop();
ros::spin();
return 0;

};

```

A.2 imitate_iiwa.cpp

```

#include "ros/ros.h"
#include "iiwa_msgs/JointPosition.h"
#include "geometry_msgs/PoseStamped.h"
#include <std_msgs/Float64.h>

iiwa_msgs::JointPosition current_joint_position, command_joint_position;
geometry_msgs::PoseStamped current_cartesian_position, command_cartesian_position;
std::string joint_position_topic, cartesian_position_topic,
    ↪ command_cartesian_position_topic, command_joint_position_topic;
double left_elbow_roll, left_elbow_pitch, left_elbow_yaw, left_shoulder_roll,
    ↪ left_shoulder_pitch, left_shoulder_yaw, right_elbow_roll, right_elbow_pitch
    ↪ , right_elbow_yaw, right_shoulder_roll, right_shoulder_pitch,
    ↪ right_shoulder_yaw;
double left_elbow_angle, left_shoulder_angle, right_elbow_angle,
    ↪ right_shoulder_angle, right_elbow_rotation, right_shoulder_rotation;
double ros_rate = 1.0;

```

```

bool isRobotConnected = false , use_right_hand = true , first = true;

void jointPositionCallback(const iiwa_msgs::JointPosition& jp)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    current_joint_position = jp;
}

void cartesianPositionCallback(const geometry_msgs::PoseStamped& ps)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    current_cartesian_position = ps;
}

void left_elbow_rollCallback(const std_msgs::Float64& ler)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    left_elbow_roll = ler.data;
}

void left_elbow_pitchCallback(const std_msgs::Float64& lep)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    left_elbow_pitch = lep.data;
}

void left_elbow_yawCallback(const std_msgs::Float64& ley)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    left_elbow_yaw = ley.data;
}

void left_shoulder_rollCallback(const std_msgs::Float64& lsr)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    left_shoulder_roll = lsr.data;
}

void left_shoulder_pitchCallback(const std_msgs::Float64& lsp)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    left_shoulder_pitch = lsp.data;
}

void left_shoulder_yawCallback(const std_msgs::Float64& lsy)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    left_shoulder_yaw = lsy.data;
}

void right_elbow_rollCallback(const std_msgs::Float64& rer)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    right_elbow_roll = rer.data;
}

```



```

}

void right_elbow_pitchCallback(const std_msgs::Float64& rep)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    right_elbow_pitch = rep.data;
}

void right_elbow_yawCallback(const std_msgs::Float64& rey)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    right_elbow_yaw = rey.data;
}

void right_shoulder_rollCallback(const std_msgs::Float64& rsr)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    right_shoulder_roll = rsr.data;
}

void right_shoulder_pitchCallback(const std_msgs::Float64& rsp)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    right_shoulder_pitch = rsp.data;
}

void right_shoulder_yawCallback(const std_msgs::Float64& rsy)
{
    if (!isRobotConnected)
        isRobotConnected = !isRobotConnected;
    right_shoulder_yaw = rsy.data;
}

int main (int argc, char** argv) {
    ros::init(argc, argv, "imitate");
    ros::NodeHandle nh("~");
    ros::AsyncSpinner spinner(1);
    spinner.start();
    nh.param("joint_position_topic", joint_position_topic, std::string("/iiwa/
        ↪ state/JointPosition"));
    nh.param("cartesian_position_topic", cartesian_position_topic, std::string(
        ↪ "/iiwa/state/CartesianPose"));
    nh.param("command_cartesian_position_topic",
        ↪ command_cartesian_position_topic, std::string("/iiwa/command/
        ↪ CartesianPose"));
    nh.param("command_joint_position_topic", command_joint_position_topic, std
        ↪ ::string("/iiwa/command/JointPosition"));
    nh.param("use_right_hand", use_right_hand, true);
    nh.param("ros_rate", ros_rate); // 0.1 Hz = 10 seconds
    ros::Rate* loop_rate_ = new ros::Rate(ros_rate);

    // Subscribers and publishers
    ros::Subscriber sub_joint_position = nh.subscribe(joint_position_topic, 1,
        ↪ jointPositionCallback);
    ros::Subscriber sub_cartesian_position = nh.subscribe(
        ↪ cartesian_position_topic, 1, cartesianPositionCallback);
    ros::Subscriber left_elbow_roll_sub = nh.subscribe("/left_elbow_roll", 1,
        ↪ left_elbow_rollCallback);

```

```

ros::Subscriber left_elbow_pitch_sub = nh.subscribe("/left_elbow_pitch", 1,
    ↪ left_elbow_pitchCallback);
ros::Subscriber left_elbow_yaw_sub = nh.subscribe("/left_elbow_yaw", 1,
    ↪ left_elbow_yawCallback);
ros::Subscriber left_shoulder_roll_sub = nh.subscribe("/left_shoulder_roll"
    ↪ , 1, left_shoulder_rollCallback);
ros::Subscriber left_shoulder_pitch_sub = nh.subscribe("/
    ↪ left_shoulder_pitch", 1, left_shoulder_pitchCallback);
ros::Subscriber left_shoulder_yaw_sub = nh.subscribe("/left_shoulder_yaw",
    ↪ 1, left_shoulder_yawCallback);
ros::Subscriber right_elbow_roll_sub = nh.subscribe("/right_elbow_roll", 1,
    ↪ right_elbow_rollCallback);
ros::Subscriber right_elbow_pitch_sub = nh.subscribe("/right_elbow_pitch",
    ↪ 1, right_elbow_pitchCallback);
ros::Subscriber right_elbow_yaw_sub = nh.subscribe("/right_elbow_yaw", 1,
    ↪ right_elbow_yawCallback);
ros::Subscriber right_shoulder_roll_sub = nh.subscribe("/
    ↪ right_shoulder_roll", 1, right_shoulder_rollCallback);
ros::Subscriber right_shoulder_pitch_sub = nh.subscribe("/
    ↪ right_shoulder_pitch", 1, right_shoulder_pitchCallback);
ros::Subscriber right_shoulder_yaw_sub = nh.subscribe("/right_shoulder_yaw"
    ↪ , 1, right_shoulder_yawCallback);

ros::Publisher pub_cartesian_command = nh.advertise<geometry_msgs::
    ↪ PoseStamped>(command_cartesian_position_topic, 1);
ros::Publisher pub_joint_command = nh.advertise<iwa_msgs:: JointPosition>(
    ↪ command_joint_position_topic, 1);

    while (ros::ok()) {
        if (isRobotConnected) {

command_joint_position = current_joint_position; command_cartesian_position =
    ↪ current_cartesian_position;

/*if(left_elbow_yaw >= 0)
left_elbow_angle = fabs(left_elbow_roll) - 1.5708;
else
left_elbow_angle = 1.5708 - fabs(left_elbow_roll);
if(left_elbow_angle > 1.13446)
left_elbow_angle = 1.13446;
command_joint_position.position.a4 = left_elbow_angle;
ROS_INFO("angle is : [%3.3f]", left_elbow_angle);
//pub_joint_command.publish(command_joint_position);

if(left_shoulder_roll >= 0)
left_shoulder_angle = 1.5708 - left_shoulder_roll;
else
left_shoulder_angle = (-3 * 1.5708) - left_shoulder_roll;
if(left_shoulder_angle < -1.74533)
left_shoulder_angle = -1.74533;
command_joint_position.position.a2 = left_shoulder_angle;
ROS_INFO("angle is : [%3.3f, %3.3f]", left_shoulder_roll, left_shoulder_angle);
pub_joint_command.publish(command_joint_position);*/
if (first){
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();
    loop_rate_-->sleep();

```

```

        loop_rate_>sleep();
        loop_rate_>sleep();
        loop_rate_>sleep();
        loop_rate_>sleep();
        loop_rate_>sleep();
        loop_rate_>sleep();
        loop_rate_>sleep();
        loop_rate_>sleep();
    }
    first = false;
}

if(right_elbow_yaw < 0)
    right_elbow_angle = fabs(right_elbow_roll) - 1.5708;
else
    right_elbow_angle = 1.5708 - fabs(right_elbow_roll);
if(right_shoulder_roll < 0)
    right_shoulder_angle = fabs(right_shoulder_roll) - 1.5708;
else
    right_shoulder_angle = (3 * 1.5708) - fabs(right_shoulder_roll);
right_elbow_rotation = fabs(right_elbow_yaw) - 1.5708;
right_shoulder_rotation = right_elbow_pitch;
//right_shoulder_rotation = -(fabs(right_shoulder_yaw) - 1.5708);

if(right_elbow_angle < -1.13446)
    right_elbow_angle = -1.13446;
if(right_elbow_angle > 1.13446)
    right_elbow_angle = 1.13446;
if(right_shoulder_angle > 1.74533)
    right_shoulder_angle = 1.74533;
if(right_shoulder_angle < -1.74533)
    right_shoulder_angle = -1.74533;

command_joint_position.position.a4 = (right_elbow_angle + right_shoulder_angle);
ROS_INFO("right_elbow_angle_is: [%3.3f]", (right_elbow_angle +
    ↪ right_shoulder_angle));

command_joint_position.position.a2 = right_shoulder_angle;
ROS_INFO("right_shoulder_angle: [%3.3f]", right_shoulder_angle);

command_joint_position.position.a3 = right_elbow_rotation;
ROS_INFO("right_elbow_rotation_is: [%3.3f]", right_elbow_rotation);

command_joint_position.position.a1 = right_shoulder_rotation;
ROS_INFO("right_shoulder_rotation_is: [%3.3f]", right_shoulder_rotation);

pub_joint_command.publish(command_joint_position);

//ROS_INFO("right_elbow_roll, pitch and yaw : [%3.3f, %3.3f, %3.3f]",
    ↪ right_elbow_roll, right_elbow_pitch, right_elbow_yaw);
//ROS_INFO("right_shoulder_roll, pitch and yaw : [%3.3f, %3.3f, %3.3f]",
    ↪ right_shoulder_roll, right_shoulder_pitch, right_shoulder_yaw);

        loop_rate_>sleep();
    }
    else {
        ROS_ERROR("Robot_is_not_connected...");
        ros::Duration(5.0).sleep(); // 5 seconds
    }
}

std::cerr<<"Stopping_spinner..."<<std::endl;

```

```

    spinner.stop();

    std::cerr<<"Bye!"<<std::endl;
    ros::spin();
    return 0;

};

```

A.3 Forward Kinematics - Denavit-Hartenberg Convention

The following are the matrices calculated in 5.8 for $A_1 \cdots A_7$.

$$A_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 \cos\frac{-\pi}{2} & \sin\theta_1 \sin\frac{-\pi}{2} & 0 \\ \sin\theta_1 & \cos\theta_1 \cos\frac{-\pi}{2} & -\cos\theta_1 \sin\frac{-\pi}{2} & 0 \\ 0 & \sin\frac{-\pi}{2} & \cos\frac{-\pi}{2} & 360 \\ 0 & 0 & 0 & 1 \end{bmatrix}; A_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 \cos\frac{\pi}{2} & \sin\theta_2 \sin\frac{\pi}{2} & 0 \\ \sin\theta_2 & \cos\theta_2 \cos\frac{\pi}{2} & -\cos\theta_2 \sin\frac{\pi}{2} & 0 \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.1)$$

$$A_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 \cos\frac{\pi}{2} & \sin\theta_3 \sin\frac{\pi}{2} & 0 \\ \sin\theta_3 & \cos\theta_3 \cos\frac{\pi}{2} & -\cos\theta_3 \sin\frac{\pi}{2} & 0 \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 420 \\ 0 & 0 & 0 & 1 \end{bmatrix}; A_4 = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 \cos\frac{-\pi}{2} & \sin\theta_4 \sin\frac{-\pi}{2} & 0 \\ \sin\theta_4 & \cos\theta_4 \cos\frac{-\pi}{2} & -\cos\theta_4 \sin\frac{-\pi}{2} & 0 \\ 0 & \sin\frac{-\pi}{2} & \cos\frac{-\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.2)$$

$$A_5 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 \cos\frac{-\pi}{2} & \sin\theta_5 \sin\frac{-\pi}{2} & 0 \\ \sin\theta_5 & \cos\theta_5 \cos\frac{-\pi}{2} & -\cos\theta_5 \sin\frac{-\pi}{2} & 0 \\ 0 & \sin\frac{-\pi}{2} & \cos\frac{-\pi}{2} & 400 \\ 0 & 0 & 0 & 1 \end{bmatrix}; A_6 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 \cos\frac{\pi}{2} & \sin\theta_6 \sin\frac{\pi}{2} & 0 \\ \sin\theta_6 & \cos\theta_6 \cos\frac{\pi}{2} & -\cos\theta_6 \sin\frac{\pi}{2} & 0 \\ 0 & \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.3)$$

$$A_7 = \begin{bmatrix} \cos\theta_7 & -\sin\theta_7 & 0 & 0 \\ \sin\theta_7 & \cos\theta_7 & 0 & 0 \\ 0 & 0 & 1 & 126 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (A.4)$$

A.4 Forward Kinematics - Translation and Rotation

The following are the matrices calculated in 5.13 for $\mathbf{d}_1^0 R_2^1 \mathbf{d}_3^2 R_4^3 \cdots \mathbf{d}_{13}^{12} R_{14}^{13} \mathbf{d}_{15}^{14}$.

$$\mathbf{d}_1^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 147.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_2^1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.5})$$

$$\mathbf{d}_3^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 212.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_4^3 = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 \\ 0 & 1 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \quad (\text{A.6})$$

$$\mathbf{d}_5^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 194.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_6^5 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.7})$$

$$\mathbf{d}_7^6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 225.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_8^7 = \begin{bmatrix} \cos\theta_4 & 0 & -\sin\theta_4 \\ 0 & 1 & 0 \\ \sin\theta_4 & 0 & \cos\theta_4 \end{bmatrix} \quad (\text{A.8})$$

$$\mathbf{d}_9^8 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.5 \\ 0 & 0 & 1 & 174.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_{10}^9 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.9})$$

$$\mathbf{d}_{11}^{10} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -60.7 \\ 0 & 0 & 1 & 225.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_{12}^{11} = \begin{bmatrix} \cos\theta_6 & 0 & \sin\theta_6 \\ 0 & 1 & 0 \\ -\sin\theta_6 & 0 & \cos\theta_6 \end{bmatrix} \quad (\text{A.10})$$

$$\mathbf{d}_{13}^{12} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 60.7 \\ 0 & 0 & 1 & 71.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Leftrightarrow R_{14}^{13} = \begin{bmatrix} \cos\theta_7 & -\sin\theta_7 & 0 \\ \sin\theta_7 & \cos\theta_7 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.11})$$

$$\mathbf{d}_{15}^{14} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 54.9 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{A.12})$$

A.5 Homogeneous Transformation code

```

geometry_msgs::Point Hf(double x, double y, double z){
geometry_msgs::Point newp; double sum; double temp [4][1];
double ht [4][4];
ht[0][0] = 0; ht[0][1] = 0; ht[0][2] = -1; ht[0][3] = 2.34;
ht[1][0] = -1; ht[1][1] = 0; ht[1][2] = 0; ht[1][3] = 0;
ht[2][0] = 0; ht[2][1] = 1; ht[2][2] = 0; ht[2][3] = 0.8;
ht[3][0] = 0; ht[3][1] = 0; ht[3][2] = 0; ht[3][3] = 1;
double oldp [4][1];
oldp[0][0] = x;
oldp[1][0] = y;
oldp[2][0] = z;
oldp[3][0] = 1;

    for (int a = 0; a < 3; a++) {
        for (int b = 0; b < 4; b++) {
            sum += ht[a][b]*oldp[b][0];
        }
        temp[a][0] = sum;
        sum = 0;
    }

newp.x = temp[0][0];
newp.y = temp[1][0];
newp.z = temp[2][0];
return newp;
}

```

A.6 Distance calculation code

```

double getDistance(geometry_msgs::Point one,
geometry_msgs::Point two){

double x1 = one.x; double y1 = one.y; double z1 = one.z;
double x2 = two.x; double y2 = two.y; double z2 = two.z;

double xSqr = double (x1 - x2) * (x1 - x2);
double ySqr = double (y1 - y2) * (y1 - y2);
double zSqr = double (z1 - z2) * (z1 - z2);

double mySqr = xSqr + ySqr + zSqr;
double myDistance = sqrt(mySqr);

return myDistance;
}

```

A.7 Position (1 obstacle)

```

geometry_msgs::Point getPose(geometry_msgs::Point one,
geometry_msgs::PoseStamped two){

if(myDistance1 >= 0.4)
factor1 = 0.1;
else if(myDistance1 >= 0.3)
factor1 = 0.2;
else if(myDistance1 >= 0.2)
factor1 = 0.3;
else if(myDistance1 >= 0.1)
factor1 = 0.35;

```

```

double xNew1 = factor1 * ((x2 - x1));
double yNew1 = factor1 * ((y2 - y1));
double zNew1 = factor1 * ((z2 - z1));

double xNew = x2 + (xNew1);
double yNew = y2 + (yNew1);
double zNew = z2 + (zNew1);

geometry_msgs::Point pose;
pose.x = xNew;
pose.y = yNew;
pose.z = zNew;

return pose;
}

```

A.8 Position (2 obstacle)

```

geometry_msgs::Point getPose(geometry_msgs::Point one,
geometry_msgs::PoseStamped two, geometry_msgs::Point three){

double factor1, factor2;

if(myDistance1 >= 0.4)
factor1 = 0.1;
else if(myDistance1 >= 0.3)
factor1 = 0.2;
else if(myDistance1 >= 0.2)
factor1 = 0.3;
else if(myDistance1 >= 0.1)
factor1 = 0.35;

if(myDistance2 >= 0.4)
factor2 = 0.1;
else if(myDistance2 >= 0.3)
factor2 = 0.2;
else if(myDistance2 >= 0.2)
factor2 = 0.3;
else if(myDistance2 >= 0.1)
factor2 = 0.35;

double xNew1 = factor1 * ((x2 - x1));
double yNew1 = factor1 * ((y2 - y1));
double zNew1 = factor1 * ((z2 - z1));

double xNew2 = factor2 * ((x2 - x3));
double yNew2 = factor2 * ((y2 - y3));
double zNew2 = factor2 * ((z2 - z3));

double xNew = x2 + ((xNew1 + xNew2) / 2);
double yNew = y2 + ((yNew1 + yNew2) / 2);
double zNew = z2 + ((zNew1 + zNew2) / 2);

geometry_msgs::Point pose;
pose.x = xNew;
pose.y = yNew;
pose.z = zNew;

return pose;
}

```

A.9 C++ code implemented

Please refer to this link <https://hg.ram.ewi.utwente.nl/> for access to all the C++ code written in this bachelor thesis or refer to the usb stick attached with the report.

Also for more information or clarification needed regarding any part of this report, kindly contact mohissa55@gmail.com.

Bibliography

- (1990), Council Directive 90/385/EEC, Technical report, European Commission.
http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/iv-diagnostic-medical-devices_en
- (1993), Council Directive 93/42/EEC, Technical report, European Commission.
<http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/medical-devices/>
- (1998), Council Directive 98/79/EC, Technical report, European Commission.
http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/iv-diagnostic-medical-devices_en
- (2006), IEC 60812:2006.
<https://webstore.iec.ch/publication/3571>
- (2006), IEC 61025:2006.
<https://webstore.iec.ch/publication/4311>
- (2012), ISO 14971 - Medical devices – Application of risk management to medical devices, Technical report, International Organisation for Standardisation.
- Agreement, G. (2015), Murab proposal.
- AmericanCancerSociety (2014), What is breast cancer?
<http://www.cancer.org/cancer/breastcancer/detailedguide/breast-cancer-what-is-breast-cancer>
- BIPM, I., I. IFCC, I. IUPAC and O. ISO (2008), The international vocabulary of metrology—basic and general concepts and associated terms (VIM), 3rd edn. JCGM 200: 2012, *JCGM (Joint Committee for Guides in Metrology)*.
- Biswas, J. and M. Veloso (2012), Depth camera based indoor mobile robot localization and navigation, in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1697–1702, ISSN 1050-4729, doi:10.1109/ICRA.2012.6224766.
- BREASTCANCER.ORG (2014), Symptoms of Breast Cancer.
http://www.breastcancer.org/symptoms/understand_bc/symptoms
- Brechon, S. (2013), A Brief History of Breast Cancer.
<http://www.maurerfoundation.org/a-brief-history-of-breast-cancer/>
- Buss, S. R. (2004), Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods, *IEEE Journal of Robotics and Automation*, vol. 17, p. 16.
- Chen, C., W. Chai, S. Wang and H. Roth (2014), A single frame depth visual gyroscope and its integration for robot navigation and mapping in structured indoor environments, in *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on*, pp. 73–78, doi:10.1109/ICARSC.2014.6849765.
- Chubb, P. (2010), Kinect Teardown: Pleo, The Dinosaur Robot Similarities.
- Comparetti, M. D. (2014), High level control of robot behavior in neurosurgery.
- Duchemin, G., P. Poignet, E. Dombre and F. Pierrot (2004), Medically safe and sound, *IEEE robotics & automation magazine*, vol. 11, pp. 46–55.
- El-laithy, R. A., J. Huang and M. Yeh (2012), Study on the use of Microsoft Kinect for robotics applications, in *Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION*, pp. 1280–1288, ISSN 2153-358X, doi:10.1109/PLANS.2012.6236985.

- Fei, B., W. S. Ng, S. Chauhan and C. K. Kwok (2001), The safety issues of medical robotics, *Reliability Engineering & System Safety*, vol. 73, pp. 183–192.
- Flacco, F., T. Kröger, A. D. Luca and O. Khatib (2012), A depth space approach to human-robot collision avoidance, in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 338–345, ISSN 1050-4729, doi:10.1109/ICRA.2012.6225245.
- Gomes, P. (2011), Surgical robotics: Reviewing the past, analysing the present, imagining the future, *Robotics and Computer-Integrated Manufacturing*, vol. 27, pp. 261–266.
- Guthart, G. and J. K. Salisbury Jr (2000), The Intuitive™ Telesurgery System: Overview and Application., in *ICRA*, pp. 618–621.
- IARC (2014), *World Cancer Report 2014*, World Health Organization.
- Jamaluddin, A. (2014), 10 Creative And Innovative Uses Of Microsoft Kinect.
<http://www.hongkiat.com/blog/innovative-uses-kinect/>
- Kazanzides, P. (2009), Safety design for medical robots, in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, pp. 7208–7211.
- Kazanzides, P., G. Fichtinger, G. D. Hager, A. M. Okamura, L. L. Whitcomb and R. H. Taylor (2008), Surgical and interventional robotics-core concepts, technology, and design [Tutorial], *IEEE Robotics & Automation Magazine*, vol. 15, pp. 122–130.
- Kinect (2009), E3 2009 : Microsoft at E3 Several Metric Tons of Press Releaseapalloza.
<http://blog.seattlepi.com/digitaljoystick/2009/06/01/e3-2009-microsoft-at-e3-several-metric-tons-of-press-releaseapalloza/>
- kobe, T. (2015), Unusual Cancers of Childhood Treatment, [Online; accessed August 12, 2016].
http://cancerinfo.tri-kobe.org/pdq/summary/english.jsp?Pdq_ID=CDR0000062878
- Kuhn, S. and D. Henrich (2007), Fast vision-based minimum distance determination between known and unknown objects, in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2186–2191, ISSN 2153-0858, doi:10.1109/IROS.2007.4399208.
- Kuka (a), KUKA LBR iiwa 14 R820 Assembly Instructions.
- Kuka (b), A revolution in robotics with a sensitive touch LBR iiwa.
- Kuka (2013), KUKA receives order for about 360 medical robots from Siemens Healthcare.
http://www.kuka-robotics.com/en/pressevents/news/NN_20130613_siemens_360_medical_robots.htm
- Maier, D., C. Lutz and M. Bennewitz (2013), Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2658–2664, ISSN 2153-0858, doi:10.1109/IROS.2013.6696731.
- Mark W. Spong, M. V. (1989), *Robot Dynamics and Control*, Wiley, ISBN 047161243X.
- Microsoft (2012), Kinect Sensor.
<https://msdn.microsoft.com/en-us/library/jj131033.aspx>
- mirror2image (2010), How Kinect depth sensor works – stereo triangulation?
<https://mirror2image.wordpress.com/2010/11/30/how-kinect-works-stereo-triangulation/>
- Mitchell, R. (2010), PrimeSense releases open source drivers, middleware that work with Kinect.
<https://www.engadget.com/2010/12/10/primesen-releases-open-source-drivers-middleware-for-kinect/>
- Ogrinc, M., T. Petrič, N. Likar, A. Gams and A. Ude (2011), Control and collision avoidance for two kuka lwr robots operated with the kinect sensor, in *20th International Workshop on*

- Robotics in Alpe-Adria-Danube Region*, pp. 173–178.
- OpenNI and ROS (2011), Kinect Accuracy, [Online; accessed August 22, 2016].
http://wiki.ros.org/openni_kinect/kinect_accuracy
- OSHA, O. S. . H. A. (2002), Industrial Robots and Robot System Safety.
https://www.osha.gov/dts/osta/otm/otm_iv/otm_iv_4.html
- Pierrot, F., E. Dombre, E. Dégoullange, L. Urbain, P. Caron, S. Boudet, J. Gariépy and J.-L. Mégnien (1999), Hippocrate: a safe robot arm for medical applications with force feedback, *Medical Image Analysis*, **vol. 3**, pp. 285–300.
- Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng (2009), ROS: an open-source Robot Operating System, in *ICRA workshop on open source software*, volume 3, p. 5.
- Seraji, H., B. Bon and R. Steele (1997), Experiments in real-time collision avoidance for dexterous 7-DOF arms, in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pp. 569–574 vol.1, doi:10.1109/ROBOT.1997.620097.
- Streiner, D. L. and G. R. Norman (2006), “Precision” and “Accuracy”: Two Terms That Are Neither, *Journal of Clinical Epidemiology*, **vol. 59**, pp. 327 – 330, ISSN 0895-4356.
- Ueki, S., T. Mouri and H. Kawasaki (2015), Collision avoidance method for hand-arm robot using both structural model and 3D point cloud, in *2015 IEEE/SICE International Symposium on System Integration (SII)*, pp. 193–198, doi:10.1109/SII.2015.7404977.
- UMM (2016), Breast Cancer Treatment and Pregnancy, [Online; accessed August 12, 2016].
<http://umm.edu/programs/cancer/healthinfo/overviews/for-patients/breast-cancer-treatment-and-pregnancy>
- Vachálek, J., L. Čapucha, P. Krasňanský and F. Tóth (2015), Collision-free manipulation of a robotic arm using the MS Windows Kinect 3D optical system, in *Process Control (PC), 2015 20th International Conference on*, pp. 96–106, doi:10.1109/PC.2015.7169945.
- Virga, S. and M. Esposito (2015), ROS indigo metapackage with ROS packages to work with the KUKA LBR IIWA R800/R820.
https://github.com/SalvoVirga/iiwa_stack
- Wang, Y., S. E. Butner and A. Darzi (2006), The developing market for medical robotics, *PROCEEDINGS-IEEE*, **vol. 94**, p. 1763.
- Wang, Y. T., C. A. Shen and J. S. Yang (2014), Calibrated kinect sensors for robot simultaneous localization and mapping, in *Methods and Models in Automation and Robotics (MMAR), 2014 19th International Conference On*, pp. 560–565, doi:10.1109/MMAR.2014.6957415.
- Zohaib, M., M. Pasha, R. A. Riaz, N. Javaid, M. Ilahi and R. D. Khan (2013), Control Strategies for Mobile Robot With Obstacle Avoidance, *ArXiv e-prints*.