

FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

Department of Mechanical and Automotive Engineering

MSc DEGREE IN

MSc Mechatronics System

Joel Eliston

ID Number: K1751269

Project Title: Baxter Robot Mirroring Human Movement

Date: 20th September 2019

Supervisor: Dr. Olga Duran

Kingston University London

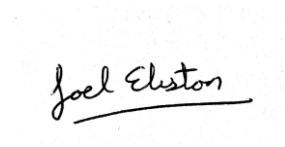
WARRANTY STATEMENT

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data or conclusion of the work performed in the project and will not be held responsible for any consequences arising out of any inaccuracies or omissions therein.

Author's Declaration

The author understands the nature of plagiarism and is aware of Kingston University's plagiarism policy.

I certify that this dissertation reports original work carried out by the author during my final year project and that all researched materials is correctly referenced in accordance with the Harvard referencing style.

A handwritten signature in black ink that reads "Joel Elston". The signature is written in a cursive style and is underlined with a single horizontal stroke.

Signature

Date: 20/09/19

ABSTRACT

In the present twenty first century, Robotics have been shortening the gap between the physical as we know and the cybernetic world. Robots are the next candidate to succeed the next industrial revolution where robotics is going to play a vital role. Robots can be presently found in every aspect of life whether it be in our homes as automated cleaning machines, site surveying drones, manufacturing industrial robots, logistics functioning robots, surgical manipulators, autonomous vehicles etc. Recent studies in computing and engineering technology has brought us closer to what we believe the future will become. With the incorporating powerful sensors like depth perception cameras, GPS systems etc., there has been a boom in the robotics industry. Modern depth cameras have given robots the ability to respond and accommodate to unadaptable environments with efficient computational system.

In 2010, Microsoft had introduced the Kinect sensor. The depth perception camera has been extremely popular in the field of robotics due to its cost efficiency and high resolution with high frame rate. In the following year, Rethink Robotics introduce the Baxter robot to the market. A semi-humanoid robot having two robotics arms with 7 degrees of freedom each, is mainly designed for the industrial field. This robot does not require hard core coding for operation, it has an easy learning functionality to perform simple tasks and also has a LCD screen that shows where it is looking and its emotional state (just like a normal person working in the industrial field).

This study incorporates the Baxter robot and Microsoft Kinect to program a mirroring system that takes the tracking data of the user and uses that data to move the Baxter robot to mirror the user's movement. This study also shows the usage of the API and SDK of the Kinect and the Baxter, all using the operating system Ubuntu to accomplish the mirror copying.

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisor, Dr. Olga Duran for giving me great support during my dissertation period. Her constant feedback and advise lead me to complete the objectives set out for this project. I would also like to thank Mr. Colin Bethell for showing me the rope on how to use Ubuntu and the basics of the Baxter robot. And also, I would like to thank Mr. Fariborz Naeini for giving me an introduction to the Baxter robot and advised me on this dissertation. Finally, I would like to thank my family and friends for their constant support and encouragement.

Table of Contents

| | | |
|-------|--|----|
| 1 | Introduction | 5 |
| 1.1 | Classifications of robots | 5 |
| 1.1.1 | Classifications based on environment | 5 |
| 1.1.2 | Classifications based on intended application field..... | 8 |
| 1.2 | Aims and Objective | 10 |
| 1.2.1 | Aim..... | 10 |
| 1.2.2 | Objective..... | 10 |
| 2 | Background and Literature Review | 11 |
| 2.1 | Hardware | 11 |
| 2.1.1 | Baxter Robot..... | 11 |
| 2.1.2 | Microsoft Kinect..... | 13 |
| 2.2 | Software..... | 15 |
| 2.2.1 | Robot Operating System..... | 15 |
| 2.2.2 | Ubuntu | 17 |
| 2.2.3 | Python..... | 19 |
| 2.3 | SDKs, APIs and Libraries | 22 |
| 2.3.1 | Baxter SDK | 24 |
| 2.3.2 | Natural User Interface | 28 |
| 2.3.3 | Libfreenect..... | 29 |
| 2.3.4 | PrimeSence NITE..... | 30 |
| 2.3.5 | OpenNI | 31 |
| 3 | Methodology | 33 |
| 3.1 | Equipment..... | 34 |
| 3.1.1 | Sensor hardware | 34 |
| 3.1.2 | Manipulator hardware | 34 |
| 3.1.3 | Computer hardware | 34 |
| 3.2 | Inverse Kinematics | 34 |
| 3.2.1 | Shoulder Roll (S0)..... | 37 |
| 3.2.2 | Shoulder Pitch (S1)..... | 37 |
| 3.2.3 | Elbow Roll (E0)..... | 37 |
| 3.2.4 | Elbow Pitch (E1) | 38 |
| 3.3 | Python code | 38 |
| 3.3.1 | Script 1 – Publer | 38 |
| 3.3.2 | Script 2 – Controller | 42 |
| 4 | Experiments..... | 44 |
| 4.1 | OpenNI Tracker test | 44 |
| 4.2 | Test using Baxter in Gazebo..... | 46 |

| | | |
|-----|--|----|
| 4.3 | Test using Baxter robot | 47 |
| 5 | Results | 48 |
| 5.1 | Result - OpenNI Tracker test..... | 48 |
| 5.2 | Result - Test using Baxter in Gazebo | 48 |
| 5.3 | Result - Test using Baxter robot..... | 48 |
| 6 | Conclusion and future works | 49 |
| 6.1 | Conclusion..... | 49 |
| 6.2 | Improvements and future works | 49 |
| | References | 51 |

List of Figures

| | |
|---|----|
| Figure 1: robotics | 5 |
| Figure 2: Classifications based on environment | 5 |
| Figure 3: Autonomous cars | 6 |
| Figure 4: Robotics vacuum cleaner | 6 |
| Figure 5: OceanOne Robot | 6 |
| Figure 6: Unmanned ground vehicle | 7 |
| Figure 7: Unmanned aerial vehicle | 7 |
| Figure 8: Classifications based on intended application field | 8 |
| Figure 9: Industrial robots | 8 |
| Figure 10: Orthoses exoskeleton | 9 |
| Figure 11: LEGO Mindstorms robot | 9 |
| Figure 12: Baxter Robot | 11 |
| Figure 13: Attention ring | 12 |
| Figure 14: Condition ring | 12 |
| Figure 15: Baxter's Arm | 12 |
| Figure 16: Working of Kinect | 13 |
| Figure 17: Microsoft Kinect | 14 |
| Figure 18: Example node structure of a PVBS system in ROS | 15 |
| Figure 19: Ubuntu server | 18 |
| Figure 20: Ubuntu desktop | 19 |
| Figure 21: Baxter SDK | 24 |
| Figure 22: Baxter Research SDK | 25 |
| Figure 23: Baxter in MoveIt | 25 |
| Figure 24: Baxter in gazebo | 27 |
| Figure 25: Natural User Interface | 28 |
| Figure 26: OpenNI | 31 |
| Figure 27: OpenNI tracker | 32 |
| Figure 28: Basic methodology | 33 |
| Figure 29: Body part mapping and joint position estimation from depth image | 33 |
| Figure 30: Human Joints | 34 |

| | |
|---|----|
| Figure 31: Baxter's joints | 35 |
| Figure 32: Co-ordinate points on the body | 36 |
| Figure 33: Vector magnitude | 39 |
| Figure 34: Vector cross product | 39 |
| Figure 35: Vector dot product | 39 |
| Figure 36: Tf data to angle part 1 | 40 |
| Figure 37: Tf data to angle part 2 | 40 |
| Figure 38: Listener and talker part 1 | 41 |
| Figure 39: Listener and talker part 2 | 41 |
| Figure 40: Script 2 – Controller part 1 | 42 |
| Figure 41: Script 2 – Controller part 2 | 42 |
| Figure 42: OpenNI tracker using Rviz | 44 |
| Figure 43: OpenNI tracker using Rviz | 45 |
| Figure 44: Test using Baxter in Gazebo | 46 |
| Figure 45: Test using Baxter robot | 47 |

1 Introduction

A machine that can execute one or more tasks automatically with precision and agility can be classified as a robot. Robots are guided with either an external controller or a controller embedded within it. The variations of types of robots can be from the basic one wheel with sensors on the bumpers to avoid obstacles to high advanced robots like the mars rover. Robots are not only in the research field, but also is being wildly used in manufacturing. Industrial robotics' field can be defined as the design, study and usage of the robotic systems for manufacturing process. Basic processes like pick and place, welding, object inspection, testing etc. all done with high precision and speed. As the years go by, the demand for robots have been sky rocketing, whether it be for simple tasks like the Roombas robot that has the functionality for cleaning domestic home to the curiosity rover created by NASA for mar exploration. There are countless ways that robots can be used for simple day to day task to high grade complex situations.

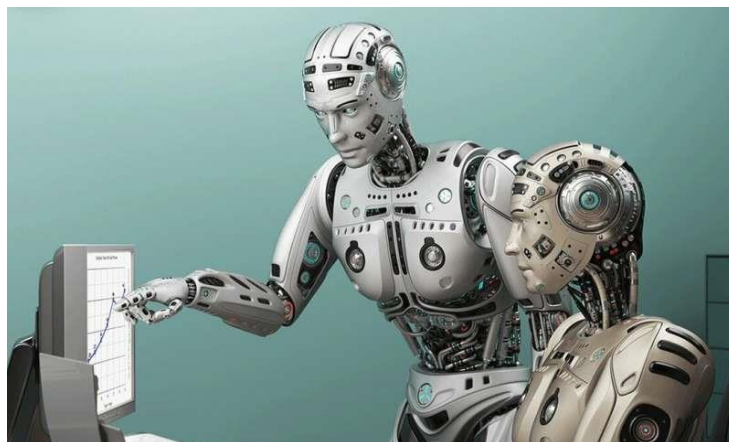


Figure 1: robotics (Phys.org, 2019)

1.1 Classifications of robots

1.1.1 Classifications based on environment

Classification of robots can be done in relation to the environment they work in. The basic distinction is mobile robots and fixed robots.

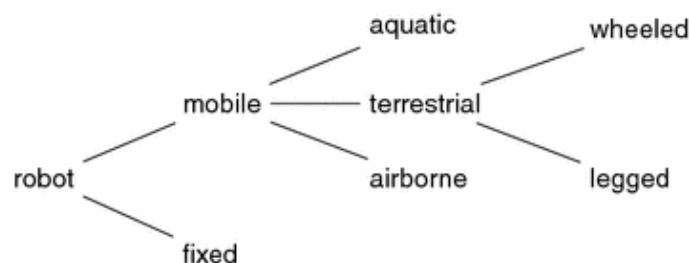


Figure 2: Classifications based on environment (Ben-Ari and Mondada, n.d.)

Fixed type robots are commonly used in the industrial field as industrial robotic manipulators that operate well in an adapted environment for the robot. Industrial based robots operate repetitive and specific operations like welding parts, reverting

operations, painting parts etc. With the advancement in technology for sensors and human interactive devices, manipulators are now increasingly being used in high precision environment like surgery.

Unlike fixed type robots, mobile robots move around ill-respect to the environment that are not specifically designed for it. These types may have an adaptive feature where deal with unfamiliar situation and environment and deal with it over time. Some examples of mobile robots are self-driving cars and robotic vacuum cleaner. (Ben-Ari and Mondada, n.d.)



Figure 3 : Autonomous cars (University, 2019)



Figure 4 : Robotics vacuum cleaner (Phys.org, 2019)

There are three main division based on the environment for mobile robots which require distinguished design styles cause there differ in the way they have to move. There are aquatic, terrestrial and airborne. (Ben-Ari and Mondada, n.d.)

Aquatic type of robots is those that are design to operate under water like the sea or the ocean for marine research. The scientist from University of Stanford have created a humanoid diving robot call OceanOne, that can travel through wrecks and coral reefs that are inaccessible to human. These types of robots can fast forward the marine research by decades. Also this type of technology can be used by the coastal guard, when an emergency situation is too risky for a person to go in.



Figure 5 : OceanOne Robot (University, 2019)

Terrestrial type robots are robots that are designed to operate on land. These are the type of robot that we see like the autonomous car, the humanoid robot like the Baxter and NAO robot, industrial robots etc. Other than these types of robots, there are some astonishing robot being researched by robotic scientists. Some scientist has developed a robot called the “Robot” which is a robot that can emit sound waves in different frequencies to map out its environment like how a bat operates. Others have developed robots for military purposes, like the Unmanned ground vehicle or UGV for short. These robot is a tactical type robot used in dangerous, impossible and inconvenient environments that is impossible for a person to operate.



Figure 6 : Unmanned ground vehicle (Ahronheim, 2019)

Coming to airborne robots, which are designed to operate in the sky. The most common example for this type of robot are drones. Drones in technical terms are unmanned aircrafts and in layman terms it is a flying robot. In today's world, UAV or unmanned aerial vehicle have many applications in the field of military, surveying, inspection and monitoring, precision agriculture, aerial imaging, computer vision, SLAM or simultaneous localization and mapping etc.



Figure 7 : Unmanned aerial vehicle (The Economic Times, 2019)

1.1.2 Classifications based on intended application field

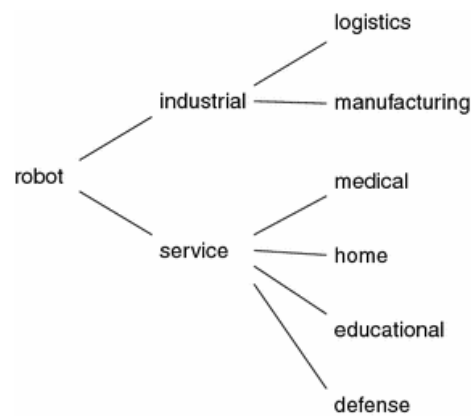


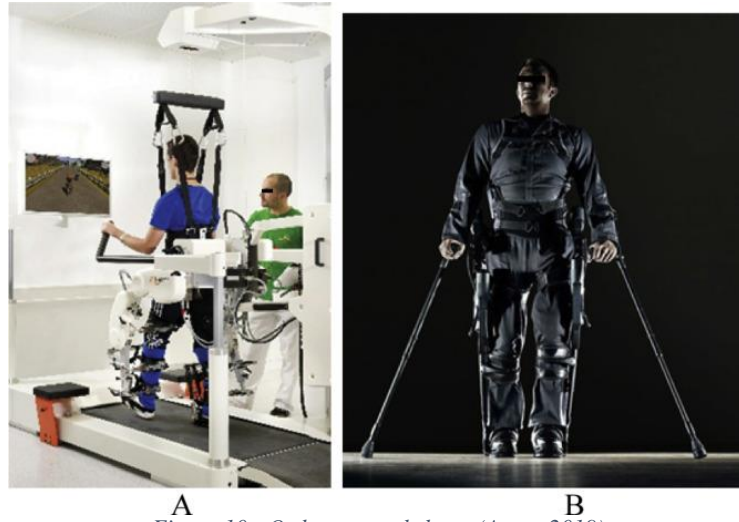
Figure 8 : Classifications based on intended application field (Ben-Ari and Mondada, n.d.)

Classification of robots can be based on the field in which they are designed for. As mentioned earlier, industrial robots operate well in adapted environment. Industrial robots are designed based on the environment they are going to work in. on the other hand, service robots are designed to assist humans in their day to day operations. These include chores at home like vacuum cleaning, defence applications like reconnaissance drones and transportation applications like self-driving vehicles. (Ben-Ari and Mondada, n.d.)



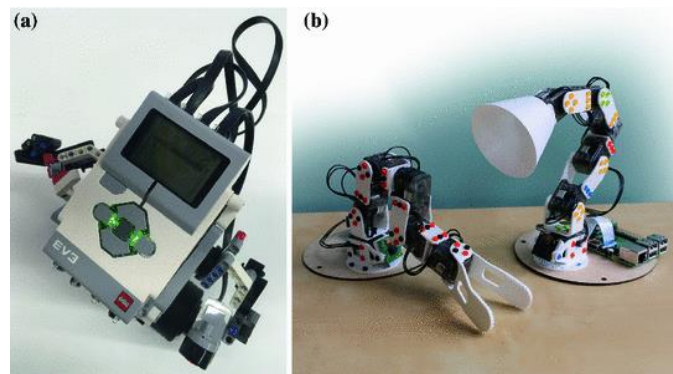
Figure 9 : Industrial robots (Ben-Ari and Mondada, n.d.)

In the medical field, these are researches being conducted to develop robots to operate basic medical checks like taking a patient's pulse, scanning for vitals, recording case notes etc. The surgical robot called "daVinci" can perform operations with just few tiny incisions with high accuracy and precision. This lead to less bleeding to occur, faster healing process and less risk for the wound to get infected. In the bio mechatronics lab in MIT, the researcher has developed a robotic prosthetic that is operated gyroscopically and can track its own position in 3 dimensional space. It has an adjust 750 times per second depending on the tracking data. Robotics exoskeleton like the Orthoses are used for paralyzed patients to move their limbs. They are also used for correcting malformations and rehabilitation purposes.



A B
Figure 10 : Orthoses exoskeleton (Anon, 2019)

Robots can be used for educational purposes to teach the current and the next generation of students. These types of robots are extensively being used in school, both for extracurricular activities and classrooms. Robots like the Thymio robot from Mobsya and the Dash robot from Wonder Workshop are specially designed educational robots which can be used by kid to learn this interesting field. Most commonly used robot for educational purposes are the Mindstorms robots developed by LEGO. This consist of a programmable “brick” which has a microcontroller that control its components like its compatible motors and sensors. Using the standard LEGO bricks, they can be customizable in many ways, may be an autonomous vehicle, an inverted pendulum etc.



(a) (b)
Figure 11 : LEGO mindstorms robot (Ben-Ari and Mondada, n.d.)

1.2 Aims and Objective

1.2.1 Aim

The aim of this study is to replicate human motion by a robotic platform using a depth camera, image processing techniques and inverse kinematics.

1.2.2 Objective

The objective of this study is to:

- To obtain the transform (tf) data of the person using Microsoft Kinect and the application program interface (API) openni.
- To perform Kinematic calculations that converts the tf tracking data to understandable data that can be used by the Baxter robot.
- Assign the right data to the correct joints of the Baxter robot to get the mimicking actions

2 Background and Literature Review

This section includes a brief description and background of all the hardware and software used in this study. The two main hardware used in this study are the Baxter robot and the Microsoft Kinect. The software used in this study includes the robotic operating system (ROS), the operating system Ubuntu, libraries, API and SDK like the OpenNI.

2.1 Hardware

The physical component of a computer systems is known as the hardware. This section includes a brief history and applications of the two hardware that are used in this study which are the Baxter robot and the Microsoft Kinect.

2.1.1 Baxter Robot

The Baxter robot was created by Rethink Robotics back in 2011 as an industrial robot. Using its two arms, the robot was designed to perform up to twelve pick and place operations per minute around with a 2*2.61 m area of reach. Baxter is approximately 3 feet tall and weighs around 165 lbs.

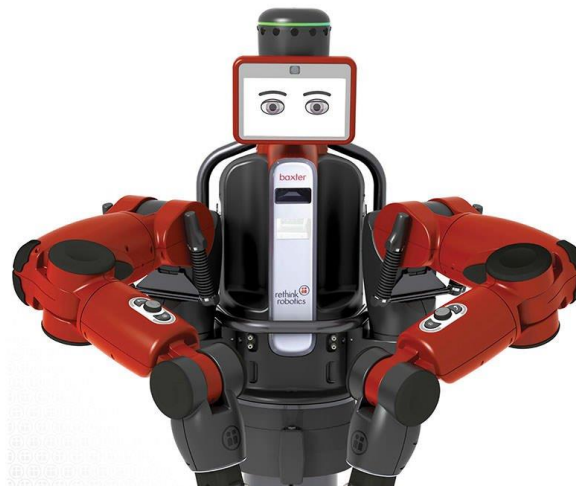


Figure 12 : Baxter Robot (Tech Explorist, 2019)

2.1.1.1 Specifications of Baxter robot

This semi-humanoid robot has an animated screen of 1024 x 600 SVGA LCD screen which is able to pan 180 degrees and also an acknowledgment feature where it nods up and down. A camera is also available on the head of the Baxter. The head of the robot also contains twelve individual sonar transducer to detect the surrounding. Each of these sonar transducer has an LED ring called the attention ring. The Condition ring above the sonar transducer give the condition of the Baxter robot. When it turns green, it means the Baxter is running a task. When turned yellow, the Baxter is confused and needs assistance by the input user. Pulsing green indicates the robot is either idle or being trained. Blinking yellow indicates it is in sleep mode and the motors are disable. Finally, blinking red shows that the Baxter has an error and needs assistance.

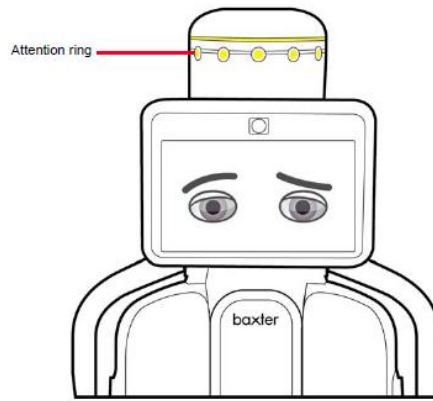


Figure 13 : Attention ring (Sdk.rethinkrobotics.com, 2019)

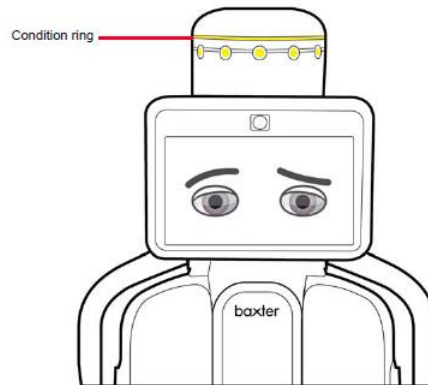


Figure 14 : Condition ring (Sdk.rethinkrobotics.com, 2019)

The arms of the Baxter robot are of length 104 cm and have 7 degrees of freedom (DOF) which are Shoulder roll (S0), Shoulder pitch (S1), Elbow roll (E0), Elbow pitch (E1), Wrist roll (W0), Wrist pitch (W1) and Wrist roll (W2).



Figure 15 : Baxter's Arm (Sdk.rethinkrobotics.com, 2019)

2.1.1.2 Applications of the Baxter robot

The Baxter robot was created to handle a large variety of repetitive operations which can replace human element in a factory floor. According to the company that created the Baxter, Rethink Robotics, these operations include:

- Unloading and loading of racks, lines etc.
- Unpacking and packing of objects from and to boxes, cases or thermoformed trays.

- Crimping and squeezing tubes
- Operating steel and heat, stamping machines and welding among other operations
- Packing products in retail blister package
- Light assembly like press-fitting plastic parts into place.
- Placing caps on bottles and jars
- Performing preliminary test of product like visual inspection of the product, checking the weight of the product and deformation checks to remove defective products.

2.1.2 Microsoft Kinect

The Microsoft Kinect which was initially known as “Project Natal”, is a motion sensing input device which is used for gaming purposes starting with the Xbox 360. Released in 2010, the Microsoft Kinect can record and understand the movement of the body and can reproduce it in the game that are compatible with the Kinect. In addition, it has face and voice recognition. These features not only make the Kinect just for game that uses your movement, it can be used as a voice activation device with video capture and face recognition capable of accomplishing a wide variety of digital tasks.

This little black box has the capability of multiple features that allow users to have maximum experience in control systems. As mentioned earlier, the Kinect is a combination of hardware and software.

2.1.2.1 Software

The developers of the software section of the Microsoft Kinect collected an enormous amount of data relating to motion capture of real moving objects in real life scenarios. Unique artificial intelligence with machine learning algorithms provided the Microsoft Kinect to map out the visual data it has gathered to models of different people using data like height, body type, gender etc. The “brain” as they call it, is the actual secret of the whole technology. It has enough intelligence to understand what visual is inputted and align this information with the stored collection of skeletal structures to know your movements. Once enough data is acquired, it gives out the output the reference data into a simple 3-dimension avatar shape. Also, the Kinect is also able to measure the distance of different points of the user’s body. This is done with the aid of a host sensor that receives the data and analysis it all in a rate of 30 times per second.

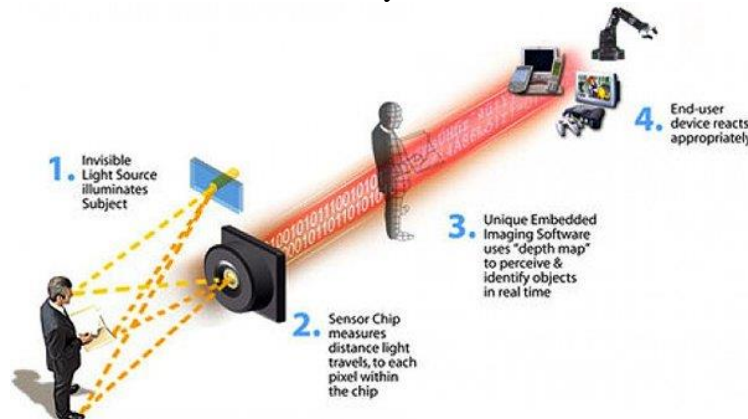


Figure 16 : Working of Kinect (Microsoft 2019)

2.1.2.2 Hardware

The hardware components of the Microsoft Kinect have three vital parts to get the motion detection and creation of the physical image of the user on the screen. These three components are:

- RGB colour camera: This camera detects the red, green and blue colour of the environment it captures as well as the facial features and body type of the user.
- Depth sensor: This consist of two sub-components which are a monochrome CMOS sensor and an infrared projector. The infrared projector sends of invisible infrared light and the CMOS sensors which is an infrared receiver that collects the reflected infrared rays. The time of flight (TOF) which is the time take for the infrared light to be reflected back is used to determine the distance of the user. This helps in creating a 3-dimensional image of the environment.
- Microphone: This section consists of four microphones that can differentiate the voices of the users from the background noise, which gives a smooth response to voice commands and voice control features.

These 3 components work together to track around 48 distinguishable points on the users' body and repeats thirty times per second

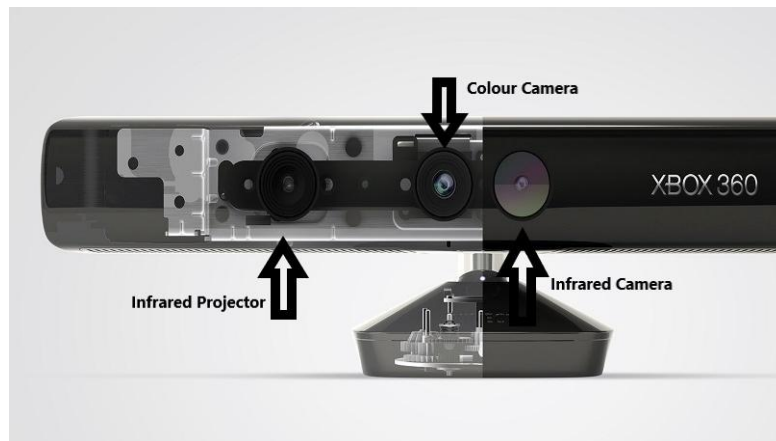


Figure 17 : Microsoft Kinect (Microsoft 2019)

2.1.2.3 Applications of Microsoft Kinect

As initially intended for gaming purposes, the Microsoft Kinect has a wide variety of applications in the field of research and science.

In the field of robotics, the Kinect is widely used as an imaging tool for its infrared camera, infrared projector and the RGB camera. To enhance the autonomous nature of robotic system, the Kinect is used as the visual instrument for mapping, localization and Simultaneous localization and mapping or SLAM for short. Other sensors other than the Kinect required for mapping have been either bulky and expensive or inaccurate and cheap. Sensors that have laser arrays are quiet heavy and cost few thousands of dollars. These sensors give a only a 2-dimensional image. Other stereo cameras available in the market which are cheap and light in weight require an enormous amount of computing power and complex algorithms to function simple tasks. Many universities use the Kinect in the research field. A group of student from UC Berkeley fixed this sensor onto a quadcopter and was successful in flying it around autonomously. Then few students from University of Bundeswehr Munich used the Microsoft Kinect on a robotic vehicle and drove it through an obstacle course. A robot that can navigate through rubble caused by a disaster like an earthquake was created by

few students from University of Warwick United Kingdom, using the Kinect as a visual and proximity sensor.

In the field of scientific research, the Microsoft Kinect is mainly used as a tool for 3-D data collection. The normally used scanning equipment was bulky and used the technology called light detection and ranging or LIDAR for short, which uses pulses of laser that can accurately measure different surfaces over couple of miles. But this equipment had a high price tag between \$10,000 to \$200,000 and also need trained professionals to operate it. Whereas the Microsoft Kinect is cheap and gives accurate data. A Ph.D. student used the Kinect in a cave in Rieperbreen Glacier in Svalbard, Norway to achieve 3-D mapping of the cave surfaces achieving the irregularity in the surface and size. The recorded data helped in the study on how the ice below flows towards the sea. Another PhD student from UC Santa Cruz used the data could be useful for missions where to deflect medium to large asteroids that threaten to impact Earth. The idea is to use one of NASA's gravity-reduced airplanes to study how a small projectile would impact a dirt pile, while the Kinect would be used to measure the three-dimensional position of objects to get data about how debris is ejected after the projectile's impact.

2.2 Software

Software is the collect of code that is installed in the computer system to aid in different operations. In this section we are going to talk about the different software used like the robotic operating system (ROS), the operating system Ubuntu and different SDKs, API and libraries used for coding.

2.2.1 Robot Operating System

Robotic operating system or ROS is an open source framework and tool set that gives functionalities of an operating system on a cluster of computers. The uses are not limited to robotics but also focuses for working with peripheral hardware. ROS contains over 2000 packages, each giving a specialised functionality. Its biggest advantage is the fact that it has one of the biggest number of tools connected to its framework.

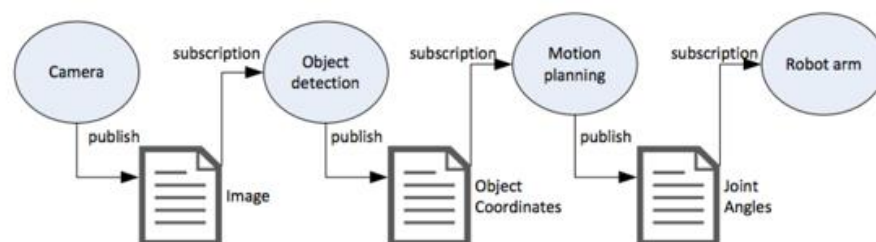


Figure 18 : Example node structure of a PVBS system in ROS (Wiki.ros.org, 2019)

The functionalities provided by ROS are device drivers, tools for testing and visualization, hardware abstraction, communications between processes running by different machines and many more. The speciality of ROS is the way the communication of the software is run, giving the luxury of designing complex software without prior knowledge on how the hardware functions. It gives a way to connect a

network of any number of process or nodes with a central hub. These nodes can be connected to that hub in various ways also can be run in multiple devices. Users can compile a complex system by assembling already solved small problems. The system is implemented in such a way that it can:

- Assembling multiple components giving multiple output into a single input to another component, thus providing solving various problems in a parallel way.
- Selecting single component from multiple components that have similar interface which gives the ability of not stopping the system for different changes.
- Creation of nodes over a link of components irrespective of where is code is being run and the implementation of Inter Process communication (IPC) systems and Remote Procedure Call (RPC) systems
- Connection of devices in different programming languages by the implementation of correct connectors to the messaging system causing easy communication for various developers to link existing modules.
- Remotely accessing the feeds from a hardware.

ROS has three divisions of concept which are the Fliesystem level, the Computation Graph level and the Community level. Also, other than the three division of ROS concept, ROS has two types of names which are the Graph Resource Names and the Package Resource Names. (Wiki.ros.org, 2019)

2.2.1.1 ROS Filesystem Level

This concept covers the ROS resources that are encountered on a disk such as:

- **Packages:** These are the main unit in the organisation of a software in ROS. A package consists of a ROS dependent library, ROS runtime process or nodes, configuration files, datasheet etc. This is considered as the most atomic build item and release item in ROS.
- **Metapackages:** These are special packages which only has a purpose to represent a group of packages that are related. Commonly used as a compatible place holder for rosbuilt stacks.
- **Message or msg type:** it is the data structure for messages transferred in ROS. Usually stored in my_package/msg/MyMessageType.msg
- **Service or srv type:** It is the request and response of a data structure for services in ROS. Usually stored in my_package/srv/MyServiceType.srv
- **Package Manifests:** This provides a metadata details of a package, which includes its version number, name, license information, description, dependencies and other meta details like exported packages.
- **Repositories:** This can be described as a collection of packages which shares a VCS system. These packages have the same version and can be released using the catkin automation tool. (Wiki.ros.org, 2019)

2.2.1.2 ROS computation graph level

The computation graph level is ROS processes that has a network that works in a peer to peer design, that processes all the data together. Nodes, messages bags, master, parameter server are the basic Computation graph concepts of ROS that provide the data.

- **Master:** the name registration and the lookup to the rest of the graph are provided by the ROS Master. The communication between the nodes like

sending and receiving of messages, discovering each other or invoke services cannot be achieved without the presence of the ROS master.

- **Bags:** the important mechanism for functions like storing sensor data are bags. They are hard to collect but essential for algorithm testing and developing.
- **Messages:** communication between nodes are done by sending “messages”. It is a simple data structure that contains typed fields. Standard data types are supported like string, integers, float, Boolean etc. Also primitive types of arrays are accepted.
- **Nodes:** In ROS, the processes that perform computations are called Nodes. For example, one node controls a sensor like an ultrasonic sensor, one node controls motor like the stepper motor, one node performs the planning of the path to be taken, one node does the localisation, one node gives a graphical representation etc. Nodes are written with the aid of the ROS library like the `rospy`.
- **Topics:** Topics are basically a name that can identify the content of a message. Messages are sent through a system of transport with a subscribe or publish semantics. Nodes send out a message by publishing it a specific “topic”. A single node could contain many concurrent subscribers or publisher and a singular node can subscribe or publish to multiple topics. Also subscribers and publishers are unaware of the existence of each other. The idea is to extract the information from both. An easy way to understand it is to think of a topic as a bus where which bus has a name and communication to a bus is possible by anyone if they have the same or right type.
- **Parameter Server:** this is a part of the master that stores data by key in a centralised location.
- **Services:** As mentioned earlier, subscribers and publishers’ models are flexible. For example, they can be one-way, many to many etc. Reply or request are done by the help of “services” which define the pair of messages. One is for reply and the other is for request. A node gives a service a name and a user can have used this service by sending a requesting message and wait for a reply from the node. (Wiki.ros.org, 2019)

2.2.2 Ubuntu

First released in 2004, Ubuntu is an operating system that was created by Canonical. It is available to the public for both professional support and the normal community. It was built on the intentions enshrined in the Ubuntu manifesto which is, it should be free of charge, the software tool must be user friendly in their language and should be customizable by user to fit their needs.

There are many advantages in using Ubuntu like:

- Unlike other Linux software, Ubuntu is stable and fast. It takes less than 60 seconds to load up.
- It is user friendly and easy to install
- The main advantage is that it is free and open source.
- It is immune to major virus unlike windows that crash easily
- The global FOSS community and Canonical provide support

- Canonical publish new version of Ubuntu once every six months and gives regular updates for free.

There are quite a few variations of Ubuntu. But the two main distinct versions are the Ubuntu Desktop and Ubuntu Server.

2.2.2.1 Ubuntu Server

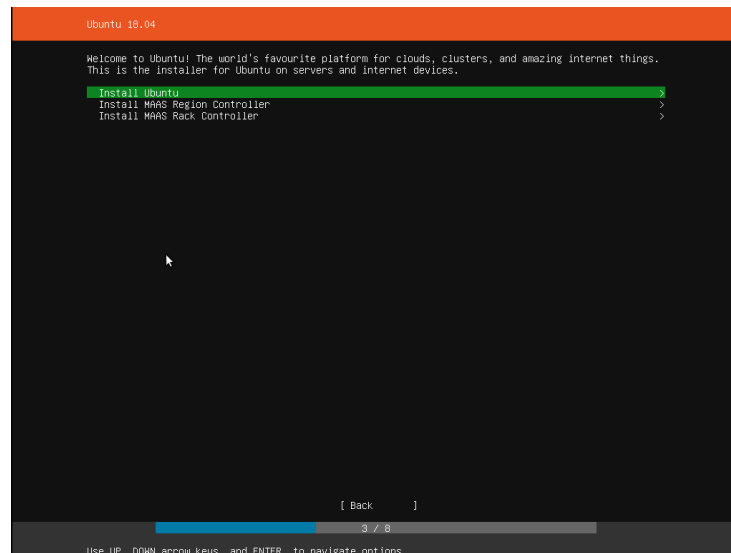


Figure 19 : Ubuntu server (Team, 2019)

The Ubuntu Server is a server version operating system created by Canonical. Some of its features include file shares, websites, containers and provide amazing cloud presence. It is mainly used by small to medium size companies that need cost effective and economical server solutions. The Ubuntu server supports most of the major architectures like IBM System z, POWER8, ARM v7, ARM64, x86 and x86-64. Some of the features are

- Email server
- FTP or File transfer protocol
- Websites
- Cloud services
- Container deployment
- Development Platform
- Print and file servers
- Database server

The Ubuntu Server requires only minimum requirements like 512 megabytes of RAM, 1 GHz CPU speed and 1 gigabytes of storage space. By default, Ubuntu comes with a wide range of software like in any other operating system like LibreOffice, Transmission and some light weight games such as chess and Sudoku. GNU General Public License is used in Ubuntu. Unlike the desktop version, the Ubuntu Server does not have a graphical user interface. Server usually use remotely managed using SSH. Ubuntu Server comes with some standard packages that focus on server requirements. Specific packages that are compatible with the server version are Bind9 and Apache2.

2.2.2.2 Ubuntu desktop



Figure 20 : Ubuntu desktop (OMG! Ubuntu!, 2019)

Ubuntu Desktop is an operating system with a graphical user interface. Even though the desktop version is an alternative to Windows or Mac, it is highly dependent on the command line, which is known as terminal for operating functions. Unlike Windows and Mac, Ubuntu uses the terminal to execute files. The desktop version has a simple and unique file system structure that is used to navigate to the file manager. The “file explorer” is user friendly and not that different compared to Mac’s Finder or Windows’ Windows Explorer. Also, there are various other applications that can be installed to replace the File explorer. Since the desktop version is an open source operating system, it is much more customizable than Mac and Windows. These may require a third party software which can be easily installed using the terminal. The desktop version comes with a “dash”, which is a tool bar located on the left side and its classic panel which is located on the top. Similar to Mac and Windows, the background is customizable and icons can be created on the desktop. But they differ in a way that, the terminal is required to do so.

The desktop version comes with many pre-installed applications and many more are stored in the repository that can be downloaded and installed using the terminal. Some of the pre-installed applications are Firefox which is Ubuntu’s default web browser, Empathy which is a chat services account manager, Movie Player which is a video and audio player, Shotwell Photo Manager which is an image manager and photo editor, Gedit which is a document editor similar to that of TextEdit in Mac and Notepad in Windows, Ubuntu One Music Store is a music browser and store which is similar to that of Spotify and iTunes and finally, Thunderbird which is an e-mail client similar to that of Outlook for Microsoft.

2.2.3 Python

“First developed in the late 1980s by Guido van Rossum, Python has advanced as an open source programming language by managing public discussion through Python Enhancement Proposals (PEPs). In 2018, van Rossum stepped down as the language’s

Benevolent Dictator For Life (BDFL), and, as officially outlined in PEP 13, a steering council was put in place to serve as the leadership of the language. The Python Software Foundation (PSF) is a 501(c)(3) non-profit corporation that holds the intellectual property rights behind the Python programming language. This includes Python version 2.1 and later, PyPI, the CPython reference implementation, and infrastructure to maintain the language. The PSF also provides grants for software craft ship and runs multiple PyCon conferences a year. (Opensource.com, 2019)

Python is a programming language. It is a high level, object oriented language with dynamic semantics. It consists of high level data structures combining with dynamic binding and dynamic typing making it very efficient for functions like Rapid Application development, as well as script writing and glue languages which can be used for the communication between existing components. Compared to other programming languages like C or java, Python has an easy to understand syntax writing and readability which can decrease the cost of program maintenance. Modules and packages are supported by Python which can help users to code in a modular structure thus helping in reusing code for other tasks.

Python comes with no complicated steps for its use which lead to more productivity. Also, this helps the debugging cycle to be amazingly fast. Debugging this language does not cause a segmentation fault when a bug or a bad input is present. Instead it provides an expectation to be raised. When an expectation can not be caught, it prints a stack trace.” The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective(Python.org, 2019)

2.2.3.1 Python Standard Library

Python provides a large variety of facilities in its standard library. It has a built in module written in C that gives basic system access like input and output of files. There are some modules which are written in python that give some standardized functionality that gives users solution to regularly occurring problems. For windows, python comes an entire standard library but for Linux based operating systems, they usually give a collection of packages which needs packaging tools of the operating system to get a few or all of the optional components.

The Library contains many types of components. It has data types that are considered as the core of the programming language such as alphabets of the English language. In python, these are defined the form of literals and places some constrains on their semantics.

The Built in function of python has a large number of functions and types built which are available to the user. Some of them are

- **Abs(x)** : This function returns the absolute value of a number whether it is an interget or a float number. If the argument has to be a complex number, the magnitude of the value is returned. (Docs.python.org, 2019)
- **Asscii(object)**: it returns a string having the printable format of an object.
- **Bin(x)**: this function converts an integer to a binary format with prefixed with “0b”. (Docs.python.org, 2019)
- **Bool ([x])**: this functions returns a Boolean value. That is, it either gives a true or a false depending on the condition. (Docs.python.org, 2019)
- **Breakpoint(*args, **kws)**: “This gives you a debugger call site. Specifically, it calls sys.breakpointhook(), passing args and kws straight through. By default,

`sys.breakpointhook()` calls `pdb.set_trace()` expecting no arguments. In this case, it is purely a convenience function so you don't have to explicitly import `pdb` or type as much code to enter the debugger. However, `sys.breakpointhook()` can be set to some other function and `breakpoint()` will automatically call that, allowing you to drop into the debugger of choice.” (Docs.python.org, 2019)

- **class `bytearray([source[, encoding[, errors]])`:** “Return a new array of bytes. The `bytearray` class is a mutable sequence of integers in the range $0 \leq x < 256$. It has most of the usual methods of mutable sequences, described in *Mutable Sequence Types*, as well as most methods that the `bytes` type has, see *Bytes and Bytearray Operations*. The optional source parameter can be used to initialize the array in a few different ways:
 - If it is a string, you must also give the encoding (and optionally, errors) parameters; `bytearray()` then converts the string to bytes using `str.encode()`.
 - If it is an integer, the array will have that size and will be initialized with null bytes.
 - If it is an object conforming to the buffer interface, a read-only buffer of the object will be used to initialize the bytes array.
 - If it is an iterable, it must be an iterable of integers in the range $0 \leq x < 256$, which are used as the initial contents of the array.” (Docs.python.org, 2019)
- **`compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1)` :**” Compile the source into a code or AST object. Code objects can be executed by `exec()` or `eval()`. source can either be a normal string, a byte string, or an AST object. Refer to the `ast` module documentation for information on how to work with AST objects. The filename argument should give the file from which the code was read; pass some recognizable value if it wasn't read from a file ('<string>' is commonly used). The mode argument specifies what kind of code must be compiled; it can be 'exec' if source consists of a sequence of statements, 'eval' if it consists of a single expression, or 'single' if it consists of a single interactive statement (in the latter case, expression statements that evaluate to something other than None will be printed). The optional arguments flags and dont_inherit control which future statements affect the compilation of source. If neither is present (or both are zero) the code is compiled with those future statements that are in effect in the code that is calling `compile()`. If the flags argument is given and dont_inherit is not (or is zero) then the future statements specified by the flags argument are used in addition to those that would be used anyway. If dont_inherit is a non-zero integer then the flags argument is it – the future statements in effect around the call to `compile` are ignored. Future statements are specified by bits which can be bitwise ORed together to specify multiple statements. The bitfield required to specify a given feature can be found as the `compiler_flag` attribute on the `_Feature` instance in the `__future__` module”. (Docs.python.org, 2019)
- **Class object:** this function returns a new object with no features. The base of all classes are the object. This function has no arguments
- **`map(function, iterable, ...)` :** “Return an iterator that applies function to every item of alterable, yielding the results. If additional alterable arguments are passed, function must take that many arguments and is applied to the

items from all alterable in parallel. With multiple alterable, the iterator stops when the shortest alterable is exhausted. For cases where the function inputs are already arranged into argument tuples, see `itertools.starmap()`.”(Docs.python.org, 2019)

- **print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)** : “Print objects to the text stream file, separated by sep and followed by end. sep, end, file and flush, if present, must be given as keyword arguments. All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by sep and followed by end. Both sep and end must be strings; they can also be None, which means to use the default values. If no objects are given, `print()` will just write end. The file argument must be an object with a `write(string)` method; if it is not present or None, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead. Whether output is buffered is usually determined by file, but if the flush keyword argument is true, the stream is forcibly flushed”. (Docs.python.org, 2019)

The built in library also has constant in its namespace. These are

- **False**: “The false value of the bool type. Assignments to False are illegal and raise a `SyntaxError`.” (Docs.python.org, 2019)
- **True** : “The true value of the bool type. Assignments to True are illegal and raise a `SyntaxError`.” (Docs.python.org, 2019)
- **None** : “The sole value of the type `NoneType`. None is frequently used to represent the absence of a value, as when default arguments are not passed to a function. Assignments to None are illegal and raise a `SyntaxError`.” (Docs.python.org, 2019)
- **NotImplemented**: “ Special value which should be returned by the binary special methods (e.g. `__eq__()`, `__lt__()`, `__add__()`, `__rsub__()`, etc.) to indicate that the operation is not implemented with respect to the other type; may be returned by the in-place binary special methods (e.g. `__imul__()`, `__iand__()`, etc.) for the same purpose. Its truth value is true.” (Docs.python.org, 2019)
- **Ellipsis**: “The same as the ellipsis literal `“...”`. Special value used mostly in conjunction with extended slicing syntax for user-defined container data types.” (Docs.python.org, 2019)
- **__debug__**: “ This constant is true if Python was not started with an `-O` option.”

2.3 SDKs, APIs and Libraries

Software Development Kit or SDK for short, is an assembly of software used for either a specific equipment or for an operating system. They are normally included in the IDE or integrated development environment, which serves as the central programming interface. The SDKs usual contain sample code which gives the user examples and tutorials for the program and libraries. These samples help the user to gain the

knowledge on how to code basic programs using the particular SDK and eventually create complex coding. A few SDK provide sample graphical functions like icons and buttons which can be implemented in the code. Most of the companies of said equipment or operating systems, wants programmers or users to develop programs using their system. SO SDK's are usually free of charge. But since every kit is different, it would take a while for the programmer to get used to it. Hence, large and descriptive documentation is also provided with the SDK. (Techterms.com, 2019)

Application Programming Interface or API is a software intermediary that provides communication between two or more applications. "An API defines functionalities that are independent of their respective implementations, which allows those implementations and definitions to vary without compromising each other. Therefore, a good API makes it easier to develop a program by providing the building blocks." (Pearlman, 2019) . One of the major reasons to use APIs is that it allows the abstractions of features between two systems. A better way to understand how an API works is by using an example. In a restaurant, we order our food to the waiter and waiter then gives the order to the chef who prepares the food. In this scenario, the waiter is the API and the customer and the chef are the two systems. The waiter sends the information from customer to the chef, which is what an API exactly does. A commonly used APIs are the Web APIs. These API are commonly known as web services, provide a common ground for web applications or other applications that connect to each other through the internet. These types of APIs are not used by the general public but rather by companies to expand their services, some types of web APIs are Representational State Transfer (REST), Remote Procedure Call (RPC) and Simple Object Access Protocol (SOAP).

In the computing world, libraries can be considered as a collection of same or similar objects that are stored which can be used for coding software like object code or source code, fonts, templates, scripts, data files etc. some of the common types of libraries are:

- "A program library is a collection of (usually) precompiled, reusable programming routines that a programmer can "call" when writing code so that the programmer doesn't have to write it. A dynamic link library (DLL) is one type of program library. Another type of program library is a class library; whose stored routines are class definitions in object-oriented programming (OOP). Graphical user interface (GUI) components such as scroll bars, buttons, and windowing routines are generally stored in a class library". (SearchSQLServer, 2019)
- "A storage library is a collection of physical storage media such as tapes or disks and a way to access them. A tape library, for example, contains tape cartridges and a mechanism that moves them into and out of the drive(s) where their content is read or updated." (SearchSQLServer, 2019)
- "A data library is the area of a data centre (a centralized area housing computer systems and equipment) where storage media are archived. Online service providers also sometimes refer to a directory on a server containing files for downloading as a data library" (SearchSQLServer, 2019)
- "A virtual library is simply the online version of the traditional library. Books and documents are made available over the Web, and may be read on line or downloaded." (SearchSQLServer, 2019)

For the interactions between the Baxter robot, the Microsoft Kinect and the user SDKs, API and libraries were used to extract data and code. These are listed below

2.3.1 Baxter SDK

Rethink Robotics provided a software development kit for the research version of the Baxter robot. Researcher of all discipline can use the research Baxter SDK to create customizable applications to run on the Baxter hardware platform. ROS is to be used to interface between the SDK and the Baxter robot.

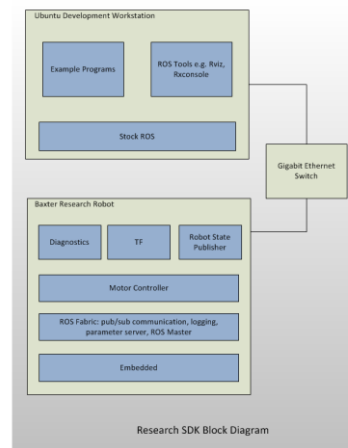


Figure 21 : Baxter SDK (GitHub, 2019)

This SDK gives a stand alone ROS master which can be connected by any workstation to connect and control the robot using various APIs of ROS. Some of the capabilities that the Baxter SDK provides are:

- Connecting to the Baxter's CPU using a secure shell (SSH). This helps users to run customized code and programs locally for full performance.
- Baxter SDK Wiki that provides user manuals, tutorials, sample codes and a section where various users can post their own code and results.
- Gazebo simulator with a full interface and also other standard ROS visualization tools like Rviz and MoveIt which gives parallel development across the world.
- A research demo mode for the Baxter Robot which allows users and programmers to run popular SDK examples to operate the Baxter's arm to navigate it without a workstation or input devices like a keyboard.

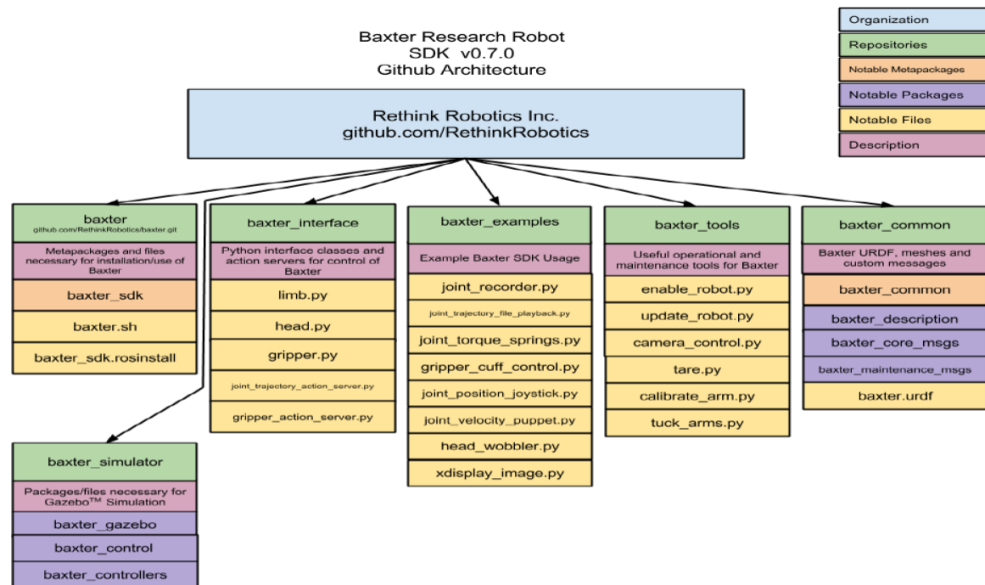


Figure 22 : Baxter Research SDK (GitHub, 2019)

This SDK was designed to work for various a scenarios thought classroom environment and laboratories. The 7 degrees of freedom arms with sensors and industry leading safety technology surges for new co-operate research and academic research in human robot interaction, AI, autonomous system, machine vision etc. The above figure shows the different elements of the software development kit which can be loaded on to the workstation to operate the Baxter robot.

As mentioned earlier, Baxter SDK provides virtual robot functions using ROS visual APIs to use the Baxter robot. They are:

2.3.1.1 MoveIt

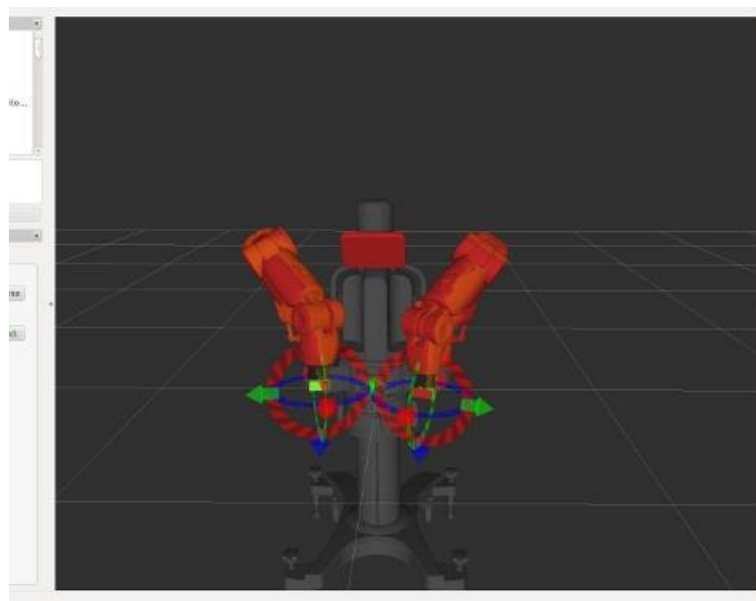


Figure 23 : Baxter in MoveIt (GitHub, 2019)

MoveIt is a ROS API which is a widely used software for manipulation. It is a robotic platform that is easy to use for the development of advanced and complex applications, evaluation of new and upgraded designs and creation of integrated products for research and development, commercial and industrial use. This gives the motion planning framework for the Baxter robot providing capabilities including kinematics and inverse kinematics like IK, FK, Jacobian calculations, motion and movement planning like OMPL, SBPL, CHOMP integrated as MoveIt! Plug-in, environment representation like collision check, robot representation, environment representation a constraint evaluation and warehouse database storage like the robot state, motion plans etc. MoveIt is configurable with the joint trajectory and action server of the Baxter robot. Baxter's SRDF or Symmetrix Remote Data Facility includes additional collision checks info, default configuration and planning groups. The planning groups for the joint of the 7 degree of freedom arms are:

Planning Groups

- both_arms
 - left_arm
 - right_arm
- left_arm
 - left_s0
 - left_s1
 - left_e0
 - left_e1
 - left_w0
 - left_w1
 - left_w2
 - left_hand
 - left_endpoint
- right_arm
 - right_s0
 - right_s1
 - right_e0
 - right_e1
 - right_w0
 - right_w1
 - right_w2
 - right_hand
 - right_endpoint
- left_hand
 - left_gripper
- right_hand
 - right_gripper

2.3.1.2 Gazebo

Gazebo is a ROS visual API that provides a 3 Dimensional dynamic simulator which has the ability to efficiently and accurately simulate a number of robots in a simple or complex indoor and outdoor environment. Similar to gaming engines, Gazebo also gives a physics simulation of a robot at a much higher degree of fidelity, communication between the program and the coder and a set of sensor interface.

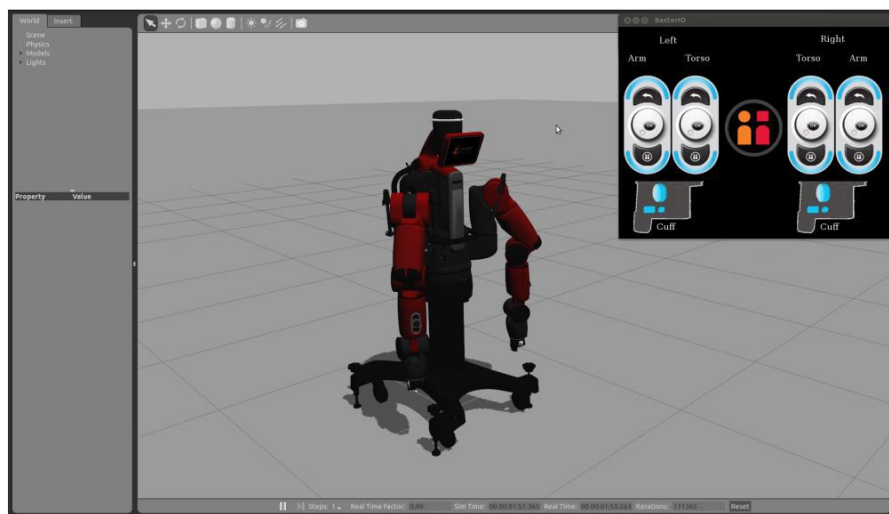


Figure 24 : Baxter in gazebo (GitHub, 2019)

The basic uses of gazebo are

- New and upgrade designing of robots.
- Testing different robotic algorithm.
- Performing regression and soak testing with real world condition.

Some of the key features are a wide variety of sensors, a large variety of libraries for robotic environments and models, multiple physics engines and a convenient graphical and programmable interface.

Baxter Research SDK adds a few increasing functionalities for the Baxter simulator in gazebo. Some of the functionalities are simple implementation of the standard electric gripper of the Baxter robot, basic functions of the arms etc. Impacts of on the simulators are:

- A new factory reset function provides a back up feature that will back up the ruser account of the programmer.
- Quick responsiveness and fidelity of the simulator of the Baxter robot.
- The source code is open sourced. That means it can be changed by the coder to what uses are need and comfortable operating the simulator.
- Repartitioning of the hard drive of the robot.
- Examples of URDF fragments which can reconfigure the electric effectors and grippers.
- Arm calibration can be done in the simulator that can give the best results in testing and automation scripts.

- Setting up objects in the robotic environment like a table or screen. This feature can be used to test various scenarios similar to real life testing. So Baxter can be used for research purposes without the need of the actual Baxter robot.
- Joint position and calculations can be achieved with the help of another ROS API called Rviz.
- Camera present on the hand of the Baxter can be calibrated. (Ugr.es, 2019)

2.3.2 Natural User Interface



Figure 25 : Natural User Interface (The Interaction Design Foundation, 2019)

A natural user interface or NUI is a system created for interaction between computer and human where the operations are done through intuitive actions related to natural, everyday human behaviour. There are many different ways in which the NUI can be operated which depends on the requirements and purpose of the user or programmer. Most of the NUI needs an intermediate device for interactions, whereas more advanced and complex NUI does not need an intermediate device. These complex NUI are either invisible to the user or so unobtrusive that they quickly seem invisible. Some of the examples for NUI are:

- Touchscreen devices
- Motion based video games like the Nintendo Wii.
- Voice recognition interface like Apple's Siri and Amazon's Alexa.
- Virtual reality simulators
- Robots responding to human motion or voice commands.

Over the years, Microsoft have accomplished many successful research into NUIs. The top researcher at Microsoft once said “exploit skills that we have acquired through a lifetime of living in the world, which minimizes the cognitive load and therefore minimizes the distraction” (The Interaction Design Foundation, 2019). Thought features like touch, voice recognition and body gestures are important factors in NUI, they will only feel natural once they are at the same skill level. NUIs are created to fit the individual person and their use of context only if they feel natural using them. Suggested by Joshua Blake, there are four main guidelines in creating and designing a NUI. They are

- Progressive learning: The user should be allowed to learn and progress on how the user interface functions. A NUI must give the opportunity for the user to start with the basics and take advance functionality step by step, in increments. One key importance is that the NUI must no overwhelm a novice user by

providing the user with too many options. Though progressive learning for novice users are gravely important, we must not forget about the expert users. They should be permitted to use their skills which was developed over time. If these experts are required to go through the long learning way like the novice users, there is a high probability of them being frustrated. So option must be provided to the user, whether they are experts or novice users to have a level of difficulty to be selected.

- **Instant Expertise:** the users' existing skills can be utilised while designing a NUI. If the user has the option to use the skills they have acquired over the years, you will be able to save them the trouble of learning the NUI from scratch. This can help users to use their existing expectations and skills to interact the NUI you have built up. There are two ways in doing so. One is to reuse the common skills that all users have. These include basic skills like speaking, writing etc. The other way is the reuse of domain specific skills. These skills are the ones that are specific to the individuals' user group. This highly depends on what type of group you are trying to build your NUI for. For example, if the NUI is built for the genre of electronics music, it is assumed that the musicians have the knowledge about different musical instruments and how to write musical notes etc.
- **Cognitive load:** if the NUI's interface is too complex and hard to interact, this will cause the user to use more cognitive load and mental effort. This will eventually make the user to lose interest in the NUI. Hence the cognitive load should be kept to a minimum. To do so, the design of the NUI in such a way that the user's simple skills and basic knowledge are primarily applied. One of the best way is to understand how objects are in the real world. The understanding of objects behaves from infancy and when NUI takes advantage of this information by the use of a graphical user interface object that behave similar to the one in the real world. When think about it, reduction of cognitive load may conflict with the instant expertise guidelines, but is still an important factor. So a balance must be made between the two.
- **Direct Interaction:** a direct co-relation between the NUI and the user can be accomplished by imitating the user's interaction with the physical world. Some of the best examples are the Microsoft Kinect and the Apple products like the iPad. Direct Interaction can be broken down into three parts. One is Directness which is basically the closeness of the user and NUI physically. The user's action happens the same time as that of the NUI or the motion of the user is followed by the motion element of the NUI. The other is the high frequency interactions. This is about the constant action and reaction between the NUI and the user. This should be designed by imitating the physical world where constant feedback is received like speed, balance, temperature etc. overwhelming the user with many feedbacks could cause cognitive load, so it is important to only show relevant information that the user desires.

2.3.3 Libfreenect

The libfreenect is a software that contains the codes that are necessary to initialize, activate and set up communication and data transfer with the Microsoft Kinect

hardware. The libfreenect has various drivers and APIs that work with cross platform operating systems like on Mac OS, Windows, Linux based systems like Ubuntu. It also supports some unique extensions and bindings for programming languages like Python, C++, C#, java etc. The OpenKinect community which is group of coders that use the Microsoft Kinect for different applications like robotics, are actively researching different possibility for setting up communications between API and OpenCV with software like MathLAB, LabVIEW and other software that are designed for networking and data socket engineering.

For this project, the libfreenect is used to initialise, connect and drive the camera component, the infrared component and the motor in the Microsoft Kinect. Libfreenect can use each comports separately depending on the task or program.

2.3.4 PrimeSense NITE

PrimeSense was a tech company that specialises in 3 dimensional sensing based in Israel. The company has produce few innovative technologies over the years. Some include Light coding technology which is a depth acquisition. This tech uses infrared light rays that on returning, the disrupted rays due to the object from the environment is recorded. This recorded data is then processed in a complex set of algorithms to triangulate and extract a 3 dimensional data.

One of their product called the PrimeSense system on a chip or SoC for short, is a CMOS image sensor in cooperated with a video sensor or an RGB camera to create a depth mapping. Primesence was then bought off by Apple on November 24 2013.

For this dissertation, we are using open of their open source system called NITE. NITE is a library used in the OpenNI SDK for the Kinect. This is a module that provides the OpenNI with features like skeleton tracking, gesture recognition, face recognition etc.

2.3.5 OpenNI

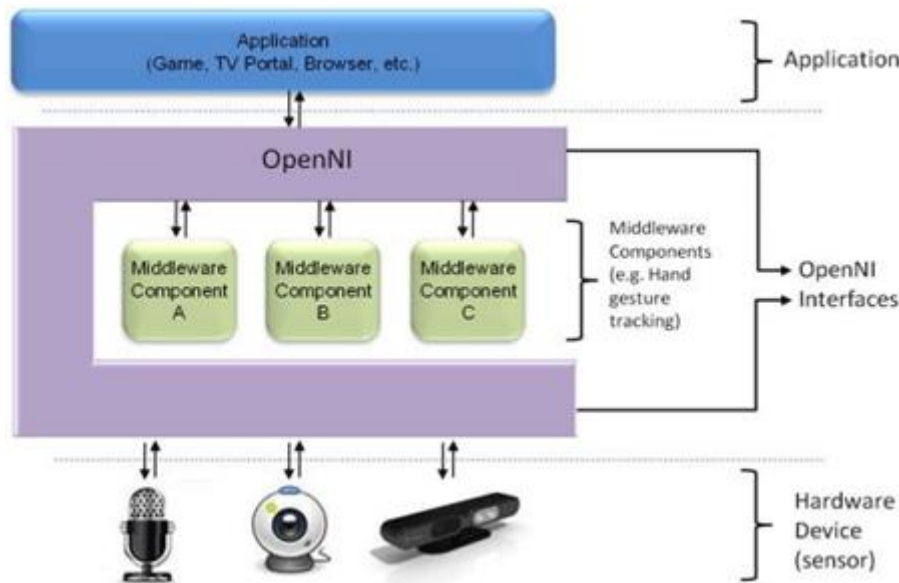


Figure 26 : OpenNI (The Interaction Design Foundation, 2019)

OpenNI is a combination of Natural user interface and Organic user interface. It is an open source framework under LGPL which was developed partially by PrimeSense. PrimeSense happens to be one of the creators of the Kinect system. This provides a standard application programming interface (API) which can be programmed using C, C++, python etc. this aids developers to write up code for application based on natural user interactions. This system is not solely for the Microsoft Kinect, it can be used for other audio and visual sensors that are compatible with it. The above figure shows the schematic architecture of the OpenNI system. To make matter simple, OpenNI allows the user to communicate to the sensors and obtain the raw data. This data is send to the middleware for analysis. This analysed data is either send to other middleware or send to applications like the Wave detector or Push Detectors. The modular architecture permits the user to communicate between different middleware and devices, complying with the API defined by OpenNI. For this study, NITE is the middleware and the Microsoft Kinect is the hardware sensor. The main package of OpenNI used in this study is the OpenNI tracker.

2.3.5.1 OpenNI tracker

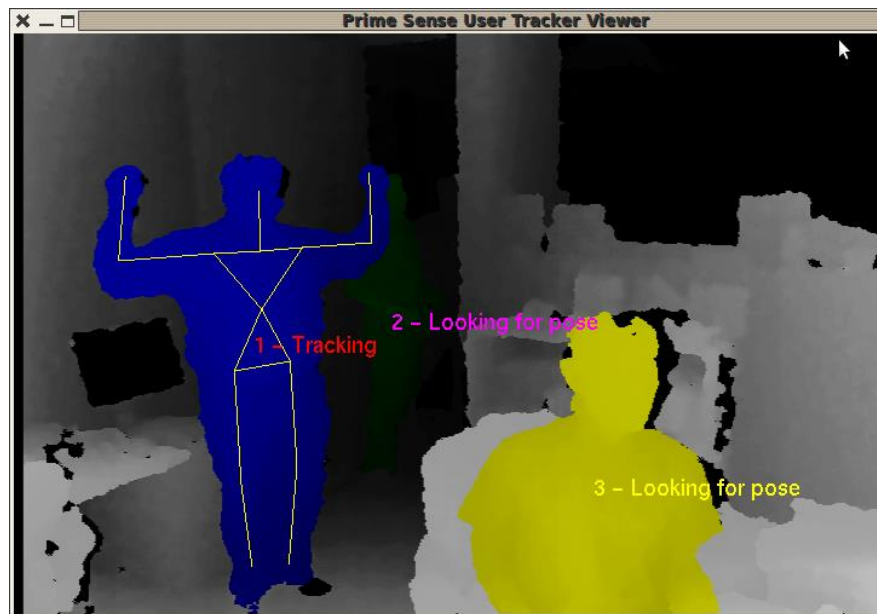


Figure 27 : OpenNI tracker (Library.isr.ist.utl.pt, 2019)

The OpenNI ROS package for skeleton tracking is the OpenNI tracker. This package is compatible with the Microsoft Kinect 360 which is connected to the PC and it has the capabilities to track up to 15 joints and parts of the human body. These joints include both hands, both elbows, head, neck, both shoulders, torso, left and right hips, both knees, and both feet. The rotation quaternions and the 3 dimensional positioning of all of these joints in space are published with a reference to the base link which is the point on the ground below the middle of the user. The OpenNI tracker analyses the transform (tf) data of the user to track the user's movements. The tf library permits the user to keep tracking the position of all the joints with respect to the base link, through a relative transformation system organised like a tree format. Thus for example, when the position of the shoulder changes, the position of the hand and the elbow get updated. There are plenty of advance functionality that is capable with transform (tf) data like stored past tf tracking data can be used to predict future movement using Kalman filtering technique. The tf produces the specification of a tree of transform between frames. Some of the data are

- map (defines the absolute reference point for all other frames)
- base_link (the point on the ground below the middle of the user)
- shoulders (at the shoulder joint)
- elbows (at the elbow joint)
- wrists (at the end of the arms)
- hands (at the end of the arms' extending sections)

There are also a few coordinate conventions to keep in mind. They will be important in getting the transformation and rotation math right.

- +y forward
- +x to the right
- +z up Angles: rotation about x, y, z axes: pitch, roll, yaw

3 Methodology

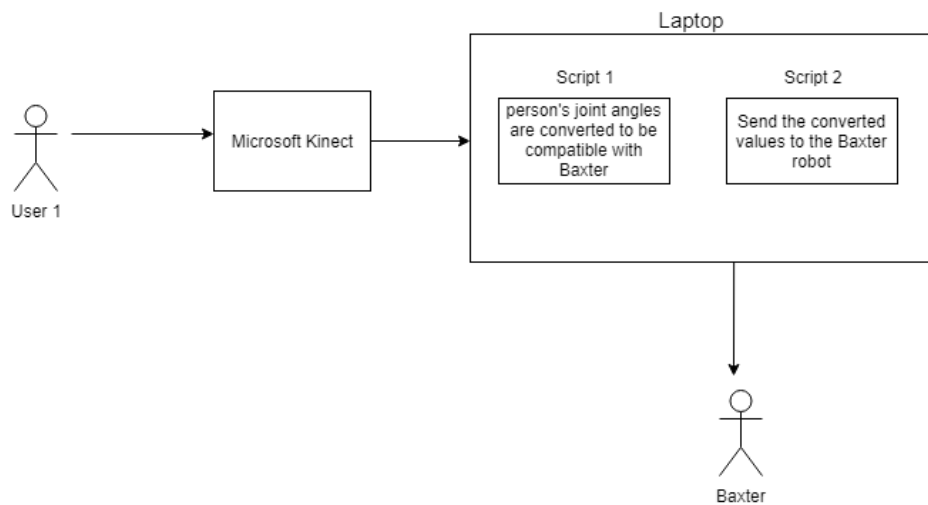


Figure 28 : Basic methodology

As mentioned in the previous section, the SDK program is used to operate the Baxter robot using the programming language python and the middleware ROS. Shown in the figure above is the basic idea on how to get the Baxter robot to mirror copy the user's arm movement. First, the Microsoft Kinect takes the depth video stream of the user and segments out or recognise each body part of the user. The OpenNI tracker which is a functionality of the SDK OpenNI, sets up tracking information of tf data (as per OpenNI tracker) for each joints with respect to the base link (which is the point on the ground below the centre of the user) as seen in the figure below. Two python scripts are used to convert the data and send it to the robot. The first script takes in only the necessary tf data needed and analyse this data to determine how many degrees should each of the joints of the robot should move in order for its arm to reach the same position as that of the user. The second script takes the angles information from the first script and allocates each joint information to the specific joints of the Baxter robot.

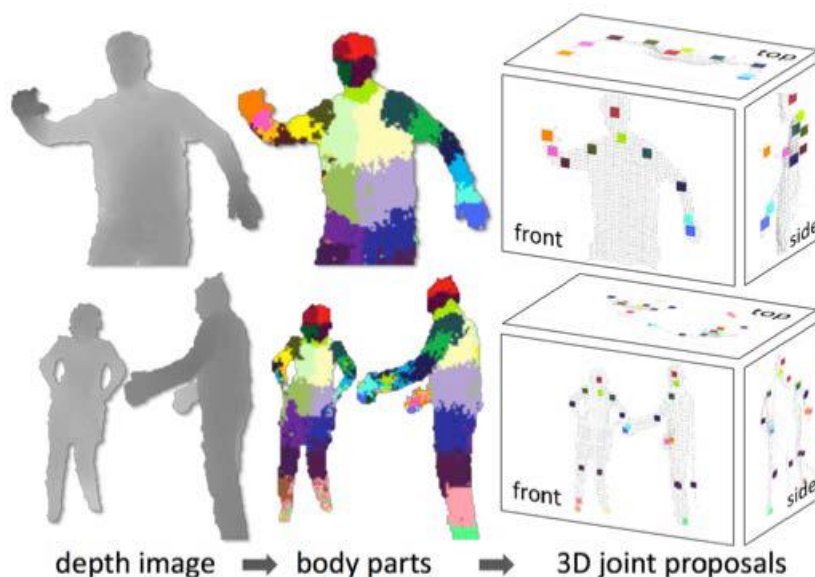


Figure 29 : Body part mapping and joint position estimation from depth image (Shotton et al 2010)

In this section, we are going to talk about all the equipment both hardware and software used to accomplish the aims and objectives of this project, the inverse kinematic calculations done to get the arm of the robot to mimic the user's arms and the python scripts developed to transfer the information from the Microsoft Kinect to the Baxter robot.

3.1 Equipment

The equipment and components used in this project were all selected to get the best result of the aims and objectives of this project. The hardware's are:

3.1.1 Sensor hardware

Microsoft Kinect for Ubuntu

3.1.2 Manipulator hardware

Baxter robot by Rethink Robotics

3.1.3 Computer hardware

Laptop running Ubuntu 16.04

3.2 Inverse Kinematics

The human arm consists of 3 major joints (excluding the joint in the finger) which are the shoulder joint, the elbow joint and the wrist joint. The shoulder joint is a ball and socket joint. This joint socket joint consist of a round shaped structure which fits on to a curved depression. This joint has all three movements which are pitch, yaw and roll all in a limited angular displacement. The elbow joint has a hinged joint similar to that of a hinge joint of a door. This type of joint has only one degree of freedom which is the pitch movement. The wrist has a saddle joint. This joint has both pitch and yaw movement which has a limited angular displacement.

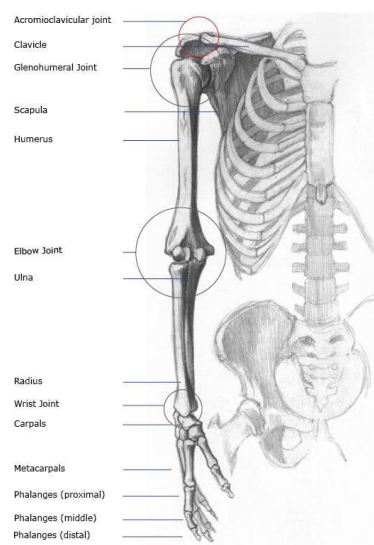


Figure 30 : Human Joints

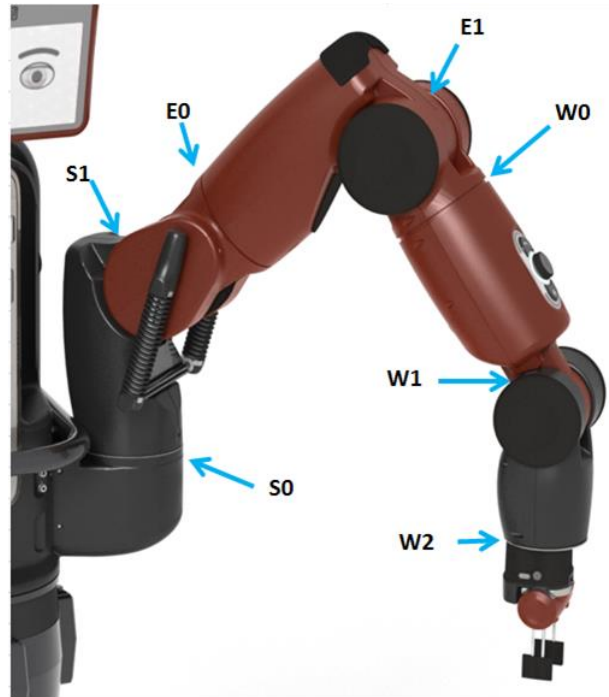


Figure 31 : Baxter's joints (Sdk.rethinkrobotics.com, 2019)

Coming to the arms of the Baxter robot which has 7 degrees of freedom, the three main joints of the arm (which are the shoulder, elbow and wrist) are split into two joints having a motor to control the movements for each joint. First the shoulder which has a shoulder roll joint (S0). This joint is a twisting joint that rotates about its axis. This joint has an angle range of 195 degrees. The shoulder pitch (S1) has a rotational joint that moves only has pitch movement (up and down movement). This joint has an angle displacement of 183 degrees. Next is the elbow joint, which has an elbow roll (E0). This joint is a twisting joint that has an angle range of 350 degrees. The elbow pitch (E1) has a rotational joint that has an angle range of 153 degrees. Now the wrist joint, which has a wrist roll (W0) which has a twisting joint. This joint has an angle range of 350.5 degrees. The wrist pitch (W1) which is a rotational joint that has an angle range of 210 degrees. And finally, the wrist roll (W2) located at the end of the arm. This joint is a twisting joint that has an angle range of 350.5 degrees.

As we can see the joint of a human arm is not similar as that of the Baxter robot. So in order to obtain the mimicking feature, we need to analyse and calculate how much angle movement is needed for each joint of the robot to reach the same position of the arm of the user. To do so we have done some inverse kinematic calculations to convert the arm movement of the user to match that of the robot

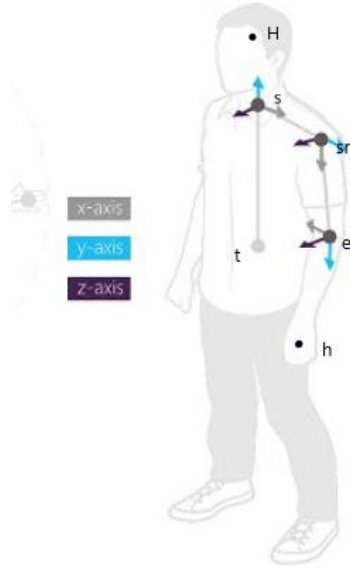


Figure 32 : Co-ordinate points on the body

From the tracking data from OpenNI tracker, we obtain the co-ordinate position of the shoulder, elbows, wrist and the hand with respect to the base like which is the point on the ground below the centre of the user. To achieve mimicking, let take the co-ordinate points of the wrist and the hand of the user as the end point to which the point of the wrist and the arm of the robot should move to. For this calculation, let us take just the right arm of the user. Let consider the following points for our calculations.

- Point between both the shoulder – s
- Point on the right shoulder – sr
- Point on the elbow – e
- Point on the hand – h
- Point on the torso – t
- And point on the head - H

Let us assume with respect to the user, the point between the shoulder (s), the head point (H) and the point on the torso (t) be stationary and points on the hand (h) as our moving point. To get the angles, we are going to use the dot product formula shown below using the points and the vectors created by the points.

$$\theta = \frac{\text{dot product between vector and vector/point}}{\text{length of vector} * \text{length of vector/point}}$$

Where $\theta = S0, S1, E0$ and $E1$

Now we will create formulae for each joint of the Baxter robot to calculate the angle it needs to move.

3.2.1 Shoulder Roll (S0)

To calculate the shoulder roll, we will need the angles created in the x-y plane of the shoulder of the Baxter robot.

To achieve that, first we get the vector between the points e and s. i.e

$$\vec{R}_{e-s} = e_{(x\ y\ z)} - s_{(x\ y\ z)}$$

Then we take the cross product of \vec{R}_{e-s} and $H_{(x\ y\ z)}$

$$\vec{V}_1 = \vec{R}_{e-s} \times H_{(x\ y\ z)}$$

We then need the to get the vector between the points sr and s

$$\vec{R}_{sr-s} = sr_{(x\ y\ z)} - s_{(x\ y\ z)}$$

Then we calculate the angle which is

$$S0 = \cos^{-1} \frac{(\vec{V}_1 \cdot \vec{R}_{sr-s})}{|\vec{V}_1| |\vec{R}_{sr-s}|}$$

3.2.2 Shoulder Pitch (S1)

To calculate the shoulder pitch, we will need the angles creates in the x- z place of the shoulder of the Baxter robot.

To achieve that, first we get the vector between the points t and H. i.e

$$\vec{R}_{t-h} = t_{(x\ y\ z)} - H_{(x\ y\ z)}$$

Then we calculate the angle which is

$$S1 = \cos^{-1} \frac{(\vec{R}_{t-h} \cdot \vec{R}_{e-s})}{|\vec{R}_{t-h}| |\vec{R}_{e-s}|}$$

3.2.3 Elbow Roll (E0)

To calculate the elbow roll, we will need the angles creates in the y-z place of the elbow of the Baxter robot

To achieve that, first we get the vector between the points h and e. i.e

$$\vec{R}_{h-e} = h_{(x\ y\ z)} - e_{(x\ y\ z)}$$

Then we take the cross product of \vec{R}_{e-s} and \vec{R}_{h-e}

$$\vec{V}_2 = \vec{R}_{e-s} \times \vec{R}_{h-e}$$

Then we calculate the angle also taking the cross product of \vec{R}_{e-s} and $H_{(x\ y\ z)}$ (\vec{V}_1) which is

$$E0 = \cos^{-1} \frac{(\vec{V}_1 \cdot \vec{V}_2)}{|\vec{V}_1| |\vec{V}_2|}$$

3.2.4 Elbow Pitch (E1)

To calculate the elbow pitch, we will need the angles created in the x- y plane of the elbow of the Baxter robot.

To achieve that, first we get the vector between the points t and H. i.e

$$\vec{R}_{h-e} = h_{(x\ y\ z)} - e_{(x\ y\ z)}$$

Then we calculate the angle also taking the vector between the points e and s (\vec{R}_{e-s}) which is

$$E1 = \cos^{-1} \frac{(\vec{R}_{e-s} \cdot \vec{R}_{h-e})}{|\vec{R}_{e-s}| |\vec{R}_{h-e}|}$$

3.3 Python code

Now that we have the formulae for the angles to be made by the joints of the Baxter robot, we have to write the code in python to get the tracking data from the OpenNI tracker, calculate the angles that need to move using the formulae and send that data to the Baxter robot. This has been achieved by using two python scripts. The first takes the tf tracking values and only uses the necessary values i.e. for this project, we take the position of the head, torso, shoulder, elbow and hand. This script also converts these values to angles that are compatible with the Baxter robot. The next script is the controller script which takes the converted values of the first script to assign the values to the correct joint of the robot and send it to the robot.

3.3.1 Script 1 – Publer

This python coded script takes in the tracking data using the `tf.TransformListener()` function to get the data of the limbs and points we need for our inverse kinematic calculations. This code is going to be divided into smaller sections for better understanding.

3.3.1.1 Vector magnitude

```
def vector_length(v):  
    return math.sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2])
```

Figure 33 : Vector magnitude

This function takes in the x, y z co-ordinates of a vector and calculates the magnitude length of that vector. The formula use is

$$length = \sqrt{vx * vx + vy * vy + vz * vz}$$

3.3.1.2 Vector cross product

```
def vector_cross_product(a, b):  
    r1 = a[1]*b[2]-b[1]*a[2]  
    r2 = a[2]*b[0]-b[2]*a[0]  
    r3 = a[0]*b[1]-b[0]*a[1]  
    return (r1, r2, r3)
```

Figure 34 : Vector cross product

This function takes in two arrays. Then using the cross product formula

$$a \times b = (r1 = ay * bz - by * az, r2 = az * bx - bz * ax, r3 = ax * by - bx * ay)$$

It then returns the value of r₁ r₂ r₃.

3.3.1.3 Vector dot product

```
def angle_query(v1, v2):  
    p = v1[0]*v2[0]+v1[1]*v2[1]+v1[2]*v2[2]  
    m1 = vector_length(v1)  
    m2 = vector_length(v2)  
    cos = p/(m1*m2)  
    return math.acos(cos)
```

Figure 35 : Vector dot product

This function takes in two vectors and takes the dot product to find the value of \cos^{-1} . The formula used are

$$p = v1 \cdot v2 = v1x * v2x + v1y * v2y + v1z * v2z$$
$$\text{inv}(\cos) = \frac{p}{\text{length of } v1 * \text{length of } v2}$$

3.3.1.4 Tf data to angle

```
def vector_cross_product(a, b):
    r1 = a[1]*b[2]-b[1]*a[2]
    r2 = a[2]*b[0]-b[2]*a[0]
    r3 = a[0]*b[1]-b[0]*a[1]
    return (r1, r2, r3)

def tf_2_angles(shoulder, shoulder_x, elbow, hand, torso, head, base, listener, limb_name):
    try:
        (b_2_s, trash) = listener.lookupTransform(base, shoulder, rospy.Time(0))
        (b_2_s_x, trash) = listener.lookupTransform(base, shoulder_x, rospy.Time(0))
        (b_2_e, trash) = listener.lookupTransform(base, elbow, rospy.Time(0))
        (b_2_hand, trash) = listener.lookupTransform(base, hand, rospy.Time(0))
        (b_2_t, trash) = listener.lookupTransform(base, torso, rospy.Time(0))
        (b_2_head, trash) = listener.lookupTransform(base, head, rospy.Time(0))

    except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
        print "tf exception"
        return ()

    b_2_s = list(b_2_s)
    b_2_s_x = list(b_2_s_x)
    b_2_e = list(b_2_e)
    b_2_hand = list(b_2_hand)
    b_2_t = list(b_2_t)
    b_2_head = list(b_2_head)

    s_2_s_x = map(operator.sub, b_2_s_x, b_2_s)
    s_2_t = map(operator.sub, b_2_t, b_2_s)
    s_2_e = map(operator.sub, b_2_e, b_2_s)
    e_2_hand = map(operator.sub, b_2_hand, b_2_e)

    head_2_t = map(operator.sub, b_2_t, b_2_head)
    my_v2 = head_2_t
```

Figure 36 : Tf data to angle part 1

```
my_v3 = vector_cross_product(s_2_e, my_v2)

my_theta_1 = angle_query(my_v3, s_2_s_x)

if limb_name == 'right':
    radius_s0 = math.pi/4-my_theta_1
else:
    radius_s0 = math.pi*3/4-my_theta_1

if mirror:
    if limb_name == 'left':
        radius_s0 = -radius_s0+math.pi/2
    else:
        radius_s0 = -radius_s0-math.pi/2

my_theta_2 = angle_query(my_v2, s_2_e)
radius_s1 = math.pi/2-my_theta_2

my_v4 = vector_cross_product(s_2_e, e_2_hand)

my_theta_3 = angle_query(my_v3, my_v4)

radius_e0 = my_theta_3
if limb_name == 'left':
    radius_e0 = -radius_e0

my_theta_4 = angle_query(s_2_e, e_2_hand)
radius_e1 = my_theta_4

return {"s0":radius_s0, "s1":radius_s1, "e0":radius_e0, "e1":radius_e1}
```

Figure 37 : Tf data to angle part 2

This function is the heart of the python code that converts the tracking data to usable angle information. This code takes in all the joint tracking data from the `tf.TransformListener()` function and assign them to appropriate variables. The variables are then distributed to respectable function to achieve the inverse kinematic functions. The value of S0, S1, E0 and E1 are achieved. The values of shoulder roll (S0) and elbow roll (E0) needs an extra step depending on which arm (left or right) the data is to be sent to. For left arms, the shoulder roll (S0) must be subtracted by 25 degrees or $\pi/4$ and elbow roll must be multiplying by -1. For right arm, shoulder roll (S0) must be added by 25 degrees and multiply by -1.

3.3.1.5 Listener and talker

```
def talker():
    global pub_list_raw, pub_list
    rospy.init_node('v8_xnode', anonymous=True)

    #pub part
    pub_left_s0 = rospy.Publisher('left_s0', Float64, queue_size=10)
    pub_left_s1 = rospy.Publisher('left_s1', Float64, queue_size=10)
    pub_left_e0 = rospy.Publisher('left_e0', Float64, queue_size=10)
    pub_left_e1 = rospy.Publisher('left_e1', Float64, queue_size=10)

    pub_right_s0 = rospy.Publisher('right_s0', Float64, queue_size=10)
    pub_right_s1 = rospy.Publisher('right_s1', Float64, queue_size=10)
    pub_right_e0 = rospy.Publisher('right_e0', Float64, queue_size=10)
    pub_right_e1 = rospy.Publisher('right_e1', Float64, queue_size=10)

    pub_list = {'left_s0':pub_left_s0,
                'left_s1':pub_left_s1,
                'left_e0':pub_left_e0,
                'left_e1':pub_left_e1,
                'right_s0':pub_right_s0,
                'right_s1':pub_right_s1,
                'right_e0':pub_right_e0,
                'right_e1':pub_right_e1}

    #tf part
    listener = tf.TransformListener()

    user_index = '1'

    for_my_left_shoulder = 'right_shoulder'+user_index
    for_my_left_shoulder_x = 'left_shoulder'+user_index
    for_my_left_elbow = 'right_elbow'+user_index
    for_my_left_hand = 'right_hand'+user_index

    for_my_right_shoulder = 'left_shoulder'+user_index
    for_my_right_shoulder_x = 'right_shoulder'+user_index
    for_my_right_elbow = 'left_elbow'+user_index
    for_my_right_hand = 'left_hand'+user_index

    head = 'head'+user_index
    torso = 'torso'+user_index
    base = 'camera_depth_frame'
```

Figure 38 : Listener and talker part 1

```
rospy.sleep(5)

rate = rospy.Rate(10) # 10hz

while not rospy.is_shutdown():

    limb_1_name = 'left'
    limb_2_name = 'right'
    if mirror == True:
        limb_1_name = 'right'
        limb_2_name = 'left'

    track_one_arm(for_my_left_shoulder,
                  for_my_left_shoulder_x,
                  for_my_left_elbow,
                  for_my_left_hand,
                  torso,
                  head,
                  base,
                  listener,
                  limb_1_name)

    track_one_arm(for_my_right_shoulder,
                  for_my_right_shoulder_x,
                  for_my_right_elbow,
                  for_my_right_hand,
                  torso,
                  head,
                  base,
                  listener,
                  limb_2_name)

    rate.sleep()

if __name__ == '__main__':
    try:
```

Figure 39 : Listener and talker part 2

This functions is where the tf tracking data is obtained from the OpenNI tracker and the converted data from the previous section is published around to a global variable called “command”. Using the `tf.TransformListener()` function which is from the OpenNI tracker’s library, all the selected tracking data is obtained and put into different variables. These variables are then sent to the other function for the inverse kinematic

calculations. The calculated values of S0, S1, E0 and E1 are then broadcasted out using a publisher function to sent it to the second script.

3.3.2 Script 2 – Controller

This is a python coded script that receives the angle data for S0, S1, E0 and E1 from the first script publer.

```
if __name__ == '__main__':
    global command
    rospy.init_node('v8_subscriber')

    rs = baxter_interface.RobotEnable(CHECK_VERSION)
    rs.enable()

    left_limb = baxter_interface.limb.Limb("left")
    right_limb = baxter_interface.limb.Limb("right")

    rospy.Subscriber("left_s0", Float64, call_back_gen('left', 0))
    rospy.Subscriber("left_s1", Float64, call_back_gen('left', 1))
    rospy.Subscriber("left_e0", Float64, call_back_gen('left', 2))
    rospy.Subscriber("left_e1", Float64, call_back_gen('left', 3))
    rospy.Subscriber("right_s0", Float64, call_back_gen('right', 0))
    rospy.Subscriber("right_s1", Float64, call_back_gen('right', 1))
    rospy.Subscriber("right_e0", Float64, call_back_gen('right', 2))
    rospy.Subscriber("right_e1", Float64, call_back_gen('right', 3))

    rospy.sleep(5)
    rate = rospy.Rate(10.0)
```

Figure 40 : Script 2 – Controller part 1

The script uses the Subscriber() function to receive the data from the publer script and assign them to different variables.

```
while not rospy.is_shutdown():
    angles = left_limb.joint_angles()
    angles['left_s0'] = command['left'][0]
    angles['left_s1'] = command['left'][1]
    angles['left_e0'] = command['left'][2]
    angles['left_e1'] = command['left'][3]
    angles['left_w0'] = 0
    angles['left_w1'] = 0
    angles['left_w2'] = 0

    #print angles
    left_limb.set_joint_positions(angles)

    angles = right_limb.joint_angles()
    angles['right_s0'] = command['right'][0]
    angles['right_s1'] = command['right'][1]
    angles['right_e0'] = command['right'][2]
    angles['right_e1'] = command['right'][3]
    angles['right_w0'] = 0
    angles['right_w1'] = 0
    angles['right_w2'] = 0
    right_limb.set_joint_positions(angles)

    rate.sleep()
```

Figure 41 : Script 2 – Controller part 2

These variables are assigned to the correct joints of the Baxter robot with respect to which side arm and the values are send to the Baxter robot for the joints to move to the particular angle.

4 Experiments

Mainly three test were conducted for this project. The first one is to get the tracking information from the OpenNI tracker using Rviz system from ROS. The next test was to conduct an experiment using the two python scripts for tracking the user's movement and get the virtual Baxter robot to move accordingly. The final test was conducted on the real Baxter robot to get the mirror movements of the user.

4.1 OpenNI Tracker test

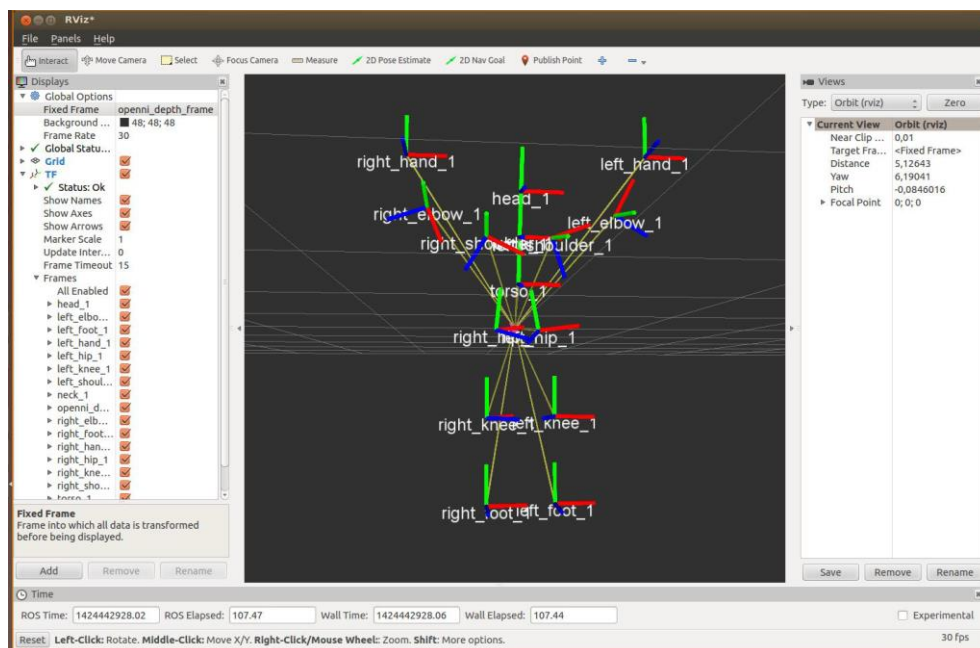


Figure 42 : OpenNI tracker using Rviz

As mentioned earlier, OpenNI tracker is an open source SDK used for the Microsoft Kinect. But after researching online, it was concluded that there were some problems related to its connection with the Kinect and also its calibration. So first, we had to check how to get the best calibrated tracker using OpenNI tracker as it was the only tracker SDK available for the Kinect 360 version.

Couple of test were conducted by using different view of the user. First was the normal test, where the user would stand in front of the camera and the tracking was enabled, but there was some discrepancy in the tracking as shown in the below figure.

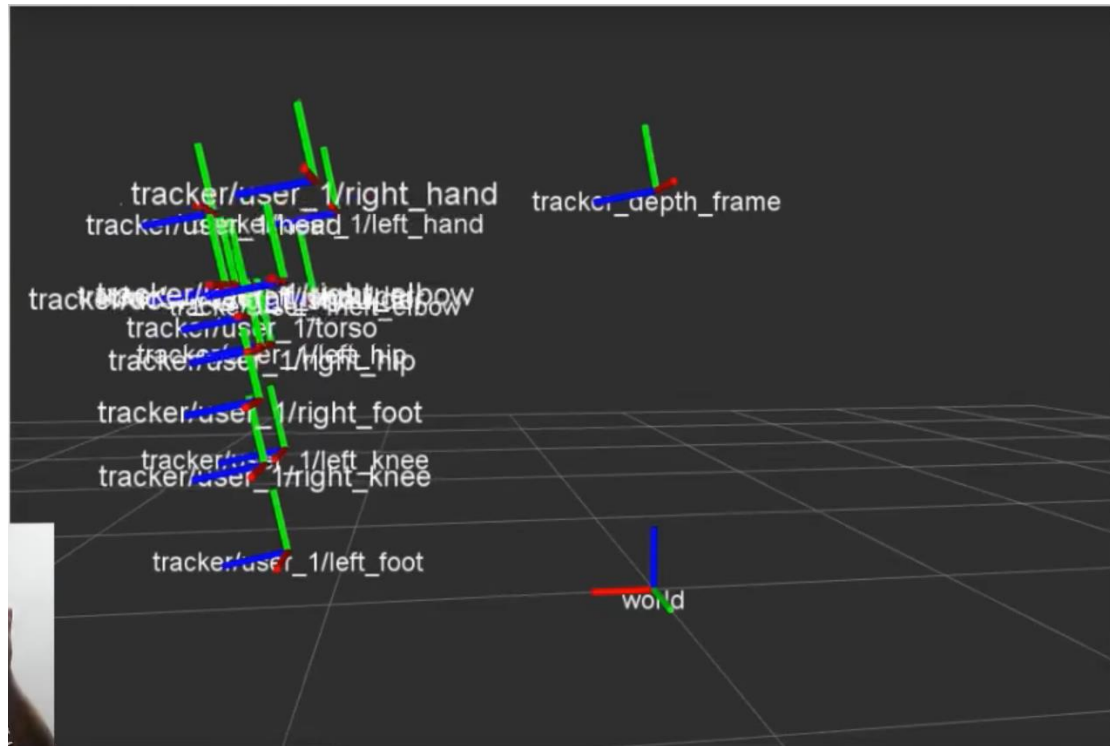


Figure 43: OpenNI tracker using Rviz

This was due to the fact that view below the knees of the user was not in frame of the camera, cause it to assume that those points are somewhere else. There were even times when tracker got confused with the point of “left foot” and “left elbow”, causing the tracking to go upside down for a few seconds.

Therefore, it was concluded that to get the optimum tracking data, the user must be approximately 2.5 meters’ way from the camera and the frame view of the camera must include the whole body of the user also well as the ground on which the user is standing to get the “base link calibrated correctly.

4.2 Test using Baxter in Gazebo

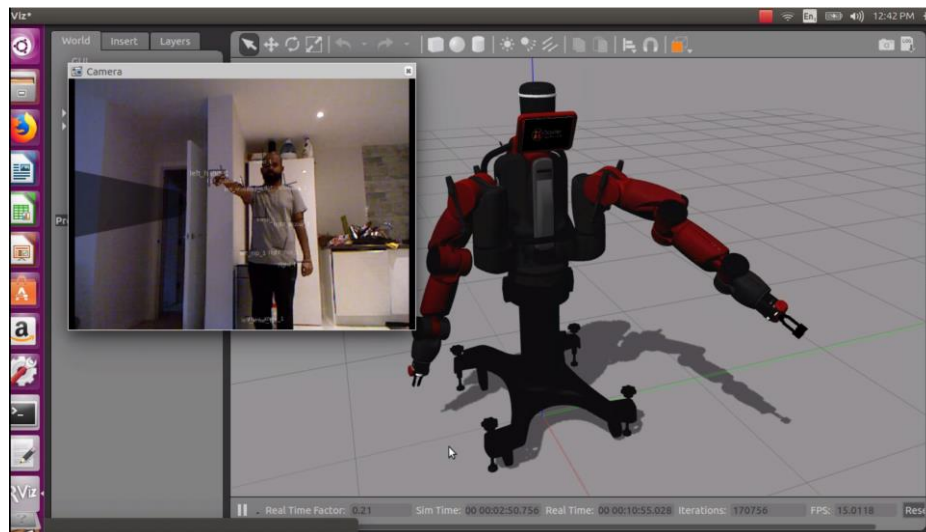


Figure 44 : Test using Baxter in Gazebo

To run the program, a couple of things need to be set up first. First the Baxter robot in gazebo needs have the IP address similar to that of the PC or laptop being used. Then five terminals need to run, which are

- The OpenNI launcher, which is used to initialize the Microsoft Kinect and access all its features.
 - The OpenNI tracker which sets up the tracking environment and starts publishing the tf tracking information of the user.
 - The Gazebo SDK which opens up the gazebo program with the Baxter plug in. this is where our virtual robot is placed.
 - The publer script that takes in the tf tracking data of the user, converts in to readable and compatible angle reading that the Baxter robot can use.
 - And the controller script which takes the angle readings from the publer script and assigns each angle value to their respective joints of the Baxter robot.
- all in the Baxter environment.

4.3 Test using Baxter robot

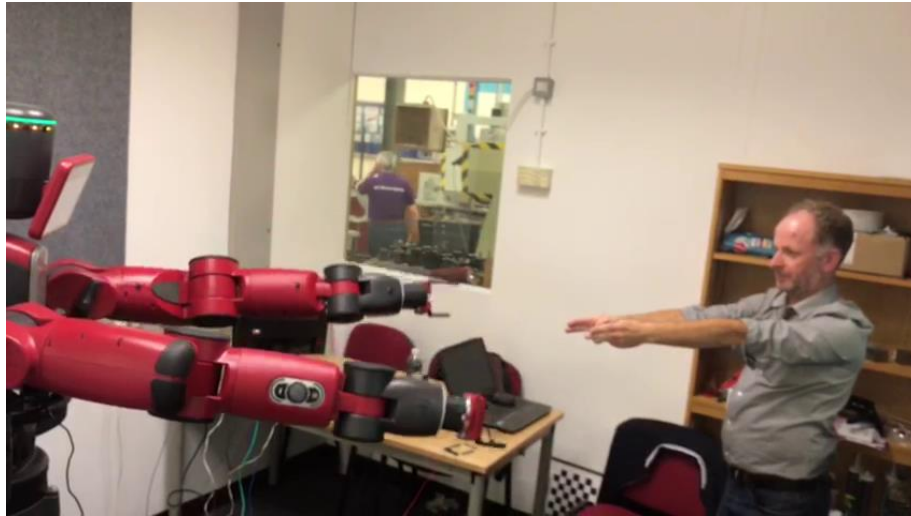


Figure 45 : Test using Baxter robot

Before conducting this test, some set ups needs to be done first. The Baxter robot is connected to the PC or laptop via Ethernet connection. The IP address of the PC or laptop must be in the same range of that of the Baxter robot. And in the Baxter SDK folder, the `baxter.sh` script needs to be edited. In this script, the IP address of the laptop or PC that is being used for this experiments, need to be given in the `baxter.sh` script. Along with the serial number of the Baxter robot being used. Once all that is done, four terminals in Ubuntu needs to be running with the following content

- The OpenNI launcher, which is used to initialize the Microsoft Kinect and access all its features.
 - The OpenNI tracker which sets up the tracking environment and starts publishing the tf tracking information of the user.
 - The publer script that takes in the tf tracking data of the user, converts in to readable and compatible angle reading that the Baxter robot can use.
 - And the controller script which takes the angle readings from the publer script and assigns each angle value to their respective joints of the Baxter robot.
- all in the Baxter environment.

5 Results

In this section, we are going to talk about the results of the three experiments conducted for this study and discussion based on the results achieved.

5.1 Result - OpenNI Tracker test

As mentioned earlier, the OpenNI is an open sourced program used for the Microsoft Kinect 360. So it does have its few problems and quirks. There were some discrepancies while conducting few tests. It was noticed that when the whole body of the user was not in frame, some tracking data like the “left foot” and “right foot” used to coincide with other tracking data like the hands. There were even times when tracker got confused with the point of “left foot” and “left elbow”, causing the tracking to go upside down for a few seconds.

Therefore, it was concluded that to get the optimum tracking data, the user must be approximately 2.5 meters’ way from the camera and the frame view of the camera must include the whole body of the user also well as the ground on which the user is standing to get the “base link calibrated correctly.

Also, OpenNI tracker could accommodate up to 8 users. If the user is set to user 1 and when the user is out of frame, its shows that user 1 is lost and when the same user comes back into frame, the tracker calibrates the user as user 2 instead of user 1.

5.2 Result - Test using Baxter in Gazebo

Test was conducted using the steps mention in the previous section for the virtual gazebo robot. The results were satisfactory. The virtual robot did connect to the publer and controller script successfully and the virtual robot did get the commands to move its joints by the angles from the controller script. But there was a lag in the virtual robot’s movements. This was due to the fact that there are 5 terminal running scripts that take up a lot of memory. Especially the terminal that run the gazebo with the Baxter plug in.

5.3 Result - Test using Baxter robot

The test was conducted using the steps mentioned in the previous section for the physical Baxter robot. The test result was good as expected. The robot receives the converted angle data from the controller script and moves its arm according to the arms of the user in real time. But there were few discrepancies noticed. First when the program is run for a period of time, like about 15 minutes or more, it was noticed that the lag was forming and the time of lag is increasing. This is due to the fact that the controller script sends out lot of data to the robot with respect to time, which eventually piles up casing the lag to occur. Also, OpenNI tracker could accommodate up to 4 users. The publer script was intended for one user (mainly user 1). So theoretically, if there are more than one person in frame, the tracking data collected in the publer script should be of the selected user. But it was noticed that the number in of the user keeps on shifting from person to person, causing the tracking data of the non-user to be collected.

6 Conclusion and future works

6.1 Conclusion

In conclusion, the goal to replicate the human motion by a robotic platform using a depth camera, image processing and inverse kinematic was achieved. The implementation of the tracking algorithm provided by the OpenNI API did manage to retrieve tracking data of the users, but had few quirks and discrepancies. Regarding the fact that the user designation keeps on changing when multiple users are detected. Also the fact that when the user is lost from the tracking frame and comes back, it designates a new name to the user rather than recognising the old user. There was the case of confused tracking data that was received when the whole body of the user was not in frame.

The inverse kinematic calculations done in this study worked quiet well in converting the tracking data to readable and compatible angles for the Baxter robot. Though there were some issues that was noticed. Mainly the fact that the strategy used for the inverse kinematics may have some faults when it comes to the height of the user. For example, if the user is below 4 feet in height and puts his or her hand all the way up, the Baxter robot may not raise its arms all the way up. This is due the fact that the distance between the coordinate points of the hand and the ground (which is take as $x=0$, $y=0$ and $z=0$ in this case) is much smaller value than the distance between the coordinate points of the arms of the Baxter robot (which is more than 6 feet tall) and the ground. The strategy is to use the location of the hand and wrist points of the user as the end point, hence casing this problem.

The two scripts used in this study worked well as expected. The first script successfully obtained the necessary tracking data from the OpenNI tracker and was able to convert it using the inverse kinematic calculations. But there was one issue when if the first script is initiated before the tracking starts, it does not receive any data from the tracker even if the tracking starts after few seconds. The second script runs smoothly without any problem. It successful enables the robot and sends the calculated angles to the respective joints in real time.

Though the overall design worked in a satisfactory level, there are few tests and improvements that can be done, which is explained in the next section.

6.2 Improvements and future works

As mentioned previously, the goal to replicate the human motion by a robotic platform using a depth camera, image processing and inverse kinematic was achieved. But will quite a few quirks and problems. This design can be improved by having a further study and different equipment. Some of the future works and improvement are:

- Accuracy test: Even though from a visual point of view, the Baxter robot seems to mimics the user correctly. But an accuracy test should have been conducted to measure the error in copying that is occurring. This can be done in few ways. One way is to use another Microsoft Kinect to get the tf tracking readings of the actual Baxter robot during operation. This data can be collected and then

compared with the tf tracking data of the user to calculate the error. Another way is to use the Optitrack Mocap system available at the Roehampton vale campus of Kingston university. With this tracking equipment, both the Baxter robot and the user's tracking information will be recorded simultaneously.

- Better tracking algorithm: Though OpenNI tracker is the only tracking API for the Microsoft Kinect 360, it is an open sourced. Meaning, it can be altering to fit the needs for this study. Furthermore, the new version of Kinect is compatible with the new OpenNI 2 API which gives better results and have fewer bugs than the old one.
- Adding low key filters: As mentioned earlier, lag occurs due to prolong usage. This is due to the accumulation of data over time. This can be avoided by using a filter that can filter out the intermediate data and only concentrate on the required data.
- Using algorithms like Kalman filter: Using addition algorithms based on machine learning like Kalman filter, that takes in the data, analyse it and predict the future movements. This can give a much smoother copying feature to this study.
- Incorporating other robots: This study mainly focusses on the Baxter robot. But this study can be applied to other robots like the NAO robot, that has more limbs like the legs which can further this study.
- Network testing: Since the communication between the Baxter robot and the laptop is done via Ethernet. Theoretically, the robot can be connected to the LAN network and can be accessed through that network. Meaning, both the user and the robot need not be in the same room. The user can be in a different floor of the building from the robot.

References

- Ackerman, E. (2012). “Kinect@Home wants to start scanning the world” Available at <http://spectrum.ieee.org/automaton/robotics/robotics-hardware/kinecthome-wants-to-start-3d-scanning-the-world> (Accessed 5th September 2019)
- Ahronheim, A. (2019). Meet RAMBOW: Israel's latest unmanned ground vehicle. [online] The Jerusalem Post | JPost.com. Available at: <https://www.jpost.com/Israel-News/Meet-RAMBOW-the-latest-unmanned-ground-vehicle-505409> [Accessed 20 Sep. 2019].
- Aldoma, A., Vincze, M., Blodow, N., Gossow, D., Gedikli, S., Rusu, R. B., Bradski, G., et al. (2019). *CAD-Model Recognition and 6DOF Pose Estimation Using 3D Cues*, 585–592.
- Alex Colon, V. (2019). The Best Robot Vacuums for 2019. [online] PCMag UK. Available at: <https://uk.pcmag.com/vacuums/74630/the-best-robot-vacuums> [Accessed 20 Sep. 2019].
- Anon, (2019). [online] Available at: https://www.researchgate.net/figure/Exoskeletons-for-gait-rehabilitation-A-The-Robotic-Orthosis-Lokomat-Image-credit_fig1_283959388 [Accessed 20 Sep. 2019].
- Araki, T., Nakamura, T., Nagai, T., Funakoshi, K., Nakano, M., & Iwahashi, N. (2011). *Autonomous acquisition of multimodal information for online object concept formation by robots*. 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1540–1547.
- Asfour, T., Regenstein, K., Azad, P., Schroder, J., Bierbaum, A., Vahrenkamp, N. & Dillmann, R. (2006) *ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control*. Available at http://mindtrans.narod.ru/pdfs/ARMAR_III.pdf (Accessed 14th September 2019)
- Azimi, M. (2011) “*Skeletal Joint Smoothing White Paper*”, Available at <http://msdn.microsoft.com/en-us/library/jj131429.aspx> (Accessed 21st July 2019)
- Attamimi, M, Mizutani, a, Nakamura, T., Nagai, T., Funakoshi, K., & Nakano, M. (2010). *Real-time 3D visual sensor for robust object recognition*. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 4560–4565.
- Benavidez, P., & Jamshidi, M. (2011). *Mobile robot navigation and target tracking system*. 2011 6th International Conference on System of Systems Engineering, 299–304. 78
- Boston Dynamics Inc (2012), “*PETMAN – BigDog gets a Big Brother*”. Available at http://www.bostondynamics.com/robot_petman.html (Accessed September 13th 2019)
- Bouguet, J. (2001). “*Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm*”. Intel Corporation, Available at http://robots.stanford.edu/cs223b04/algo_affine_tracking.pdf
- Bruce, J., Balch, T., & Veloso, M. (2000). *Fast and inexpensive color image segmentation for interactive robots*. Proceedings. 2000 IEEE/RSJ International

Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113), 3, 2061–2066.

- Buckley, M., Vaidyanathan, R., & Mayol-Cuevas, W. (2011). *Sensor suites for assistive arm prosthetics*. 2011 24th International Symposium on Computer-Based Medical Systems (CBMS), 1–6.
- Chaoui, H., & Gueaieb, W. (2007). *Type-2 Fuzzy Logic Control of a Flexible-Joint Manipulator*. Journal of Intelligent and Robotic Systems, 51(2), 159–186.
- Chaumette, F., & Hutchinson, S. (2007). *Visual servo control, Part II: Advanced approaches*. IEEE Robotics & Automation, (March), 109–118. Retrieved from <http://hal.archives-ouvertes.fr/inria-00350638/>
- Chaumette, F., Hutchinson, S. (2006). *Visual Servo Control Part I : Basic Approaches*. IEEE Robotics & Automation, (December), 82–90. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4015997
- Cheng, H., Sun, Z., & Zhang, P. (2011). *Imirok: Real-time imitative robotic arm control for home robot applications*. 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 360–363.
- Ciocarlie, M., Pantofaru, C., Hsiao, K., Bradski, G., Brook G. and Dreyfuss, E. “A Side of Data with My Robot: Three Datasets for Mobile Manipulation in Human Environments”, IEEE Robotics & Automation Magazine Special Issue: Towards a WWW for Robots, 18(2), 2011
- Cootes, T. (2000). *An introduction to active shape models*. Image Processing and Analysis, 223–248. Available at [http://person.hst.aau.dk/06gr1088d/artikler/Pdf/Cootes Introduction.pdf](http://person.hst.aau.dk/06gr1088d/artikler/Pdf/Cootes%20Introduction.pdf) (Accessed 10 May 2019)
- DARPA (2012), “*DARPA Robotics Challenge (DARPA-BA-12-39)*”, Available at <https://www.fbo.gov/utills/view?id=74d674ab011d5954c7a46b9c21597f30> (Accessed June 15th 2019)
- Devereux, D., Mitra, B., Holland, O., & Diamond, A. (2012). *Using the Microsoft Kinect to Model the Environment of an Anthropomimetic Robot*. Biomechanics / 752: Robotics.
- Diankov, R., Ueda, R., Okada, K., & Saito, H. (2011). *COLLADA : An Open Standard for Robot File Formats Introduction*. ECCE (2012). *ECCE Robot Project Description*, Available at <http://eccerobot.org/home/project/description> (Accessed 13th September 2019)
- Docs.python.org. (2019). Built-in Functions — Python 3.7.4 documentation. [online] Available at: <https://docs.python.org/3/library/functions.html> [Accessed 20 Sep. 2019].
- Federal Business Opportunities (2012), “*Sole Source Intent Notice for Humanoid Robot Systems for the DARPA Robotics Challenge Program*”, Available online at <https://www.fbo.gov/spg/ODA/DARPA/CMO/DARPA-SN-12-35/listing.html> (Accessed 13th September 2019)

- Fiala, M., & Ufkes, A. (2011). *Visual Odometry Using 3-Dimensional Video Input*. 2011 Canadian Conference on Computer and Robot Vision, 86–93.
- GitHub. (2019). RethinkRobotics/sdk-docs. [online] Available at: <https://github.com/RethinkRobotics/sdk-docs/wiki/Baxter-Research-Software-Developers-Kit-%28SDK%29> [Accessed 20 Sep. 2019].
- Gratal, X., Romero, J., Bohg, J., & Kragic, D. (2012). *Visual servoing on unknown objects*. *Mechatronics*, 22(4), 423–435.
- Hesseldahl, A., (2002) “Say Hello To Asimo”, Available at <http://www.forbes.com/2002/02/21/0221tentech.html> (Accessed 13th September 2019)
- Hinchman, W., (2011) “Kinect for Windows SDK beta vs. OpenNI”. Available at <http://labs.vectorform.com/2011/06/windows-kinect-sdk-vs-openni-2/> (Accessed 14th September 2019)
- Hirata, T., Nakamura, T., Kato, R., Morishita, S., & Yokoi, H. (2011). *Development of mobile controller for EMG prosthetic hand with tactile feedback*. 2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 110–115.
- Hutchinson, S., Hager, G. D., & Corke, P. I. (1996). *A tutorial on visual servo control*. *IEEE Transactions on Robotics and Automation*, 12(5), 651–670.
- Kalal, Z., Mikolajczyk, K., Matas, J., & Republic, C. (2010). Forward-Backward Error : Automatic Detection of Tracking Failures, 23–26.
- Library.isr.ist.utl.pt. (2019). openni_tracker. [online] Available at: http://library.isr.ist.utl.pt/docs/roswiki/openni_tracker.html [Accessed 20 Sep. 2019].
- Liu, Y. (2011) *Precision of the Kinect sensor*. Retrieved from http://www.ros.org/wiki/openni_kinect/kinect_accuracy (Accessed 6th July 2012)
- Marchand, É., & Chaumette, F. (2005). *Feature tracking for visual servoing purposes*. *Robotics and Autonomous Systems*, 52(1), 53–70.
- Matuszek, C., Mayton, B., Aimi, R., Deisenroth, M. P., Bo, L., Chu, R., Kung, M., et al. (2011). *Gambit: An autonomous chess-playing robotic system*. 2011 IEEE International Conference on Robotics and Automation, 4291–4297.
- Mavridis, N., Tsamakos, A., Giakoumidis, N., Baloushi, H., Ashkari, S. , Shamsi, M. and Kaabi, A. “Steps towards affordable android telepresence,” in Proc. HRI Workshop Social Robot. Telepresence, Lausanne, Switzerland, 2011.
- Nakamura, T. (2011), "Real-time 3-D object tracking using Kinect sensor," *Robotics and Biomimetics (ROBIO)*, 2011 IEEE International Conference on , vol., no., pp.784-788, 7-11 Dec. 2011
- Mihelich, P., & Konolige, K. (2012). *Kinect Calibration*. Retrieved from http://www.ros.org/wiki/kinect_calibration/technical (Accessed 14th September 2012)
- Muja, M., Rusu, R. B., Bradski, G., & Lowe, D. G. (2011). *REIN - A fast, robust, scalable REcognition INfrastructure*. 2011 IEEE International Conference on Robotics and Automation, 2939–2946.

- Ng, R. (2006). *Digital Light Field Photography*. Retrieved from <http://www.lytro.com/renng-thesis.pdf> (Accessed 11th September 2019)
- OMG! Ubuntu!. (2019). Ubuntu adds 'Minimal Install' option to installer - OMG! Ubuntu!. [online] Available at: <https://www.omgubuntu.co.uk/2018/02/ubuntu-18-04-minimal-install-option> [Accessed 20 Sep. 2019].
- Opensource.com. (2019). What is Python?. [online] Available at: <https://opensource.com/resources/python> [Accessed 20 Sep. 2019].
- Pearlman, S. (2019). What are APIs and how do APIs work?. [online] MuleSoft Blog. Available at: <https://blogs.mulesoft.com/biz/tech-ramblings-biz/what-are-apis-how-do-apis-work/> [Accessed 20 Sep. 2019].
- Phys.org. (2019). How to prevent the 'robot apocalypse' from ending labor as we know it. [online] Available at: <https://phys.org/news/2019-03-robot-apocalypse-labor.html> [Accessed 20 Sep. 2019].
- Python.org. (2019). What is Python? Executive Summary. [online] Available at: <https://www.python.org/doc/essays/blurb/> [Accessed 20 Sep. 2019].
- Rakprayoon, P., Ruchanurucks, M., & Coundoul, A. (2011). *Kinect-based obstacle detection for manipulator*. 2011 IEEE/SICE International Symposium on System Integration (SII), 68–73.
- Schnabel, R., Wessel, R., Wahl, R. and Klein, R. (2007) “Shape Recognition in 3D Point-Clouds”, Proceedings of The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008, UNION Agency-Science Press, Feb. 2008
- Sarabia, M., Ros, R., & Demiris, Y. (2011). *Towards an open-source social middleware for humanoid robots*. 2011 11th IEEE-RAS International Conference on Humanoid Robots, 670–675.
- SearchSQLServer. (2019). What is library? - Definition from WhatIs.com. [online] Available at: <https://searchsqlserver.techtarget.com/definition/library> [Accessed 20 Sep. 2019].
- Shi, J., & Tomasi, C. (1994). *Good features to track*. IEEE Proceedings CVPR, 593–600
- Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011), *Real-Time Human Pose Recognition in Parts from a Single Depth Image*, in CVPR, IEEE, June 2011
- Smith, C., Karayiannidis, Y., Nalpantidis, L., Gratal, X., Qi, P., Dimarogonas, D.V., Kragic, D. (2012). *Dual Arm Manipulation: a Survey*. Robotics and Autonomous Systems, 60(10), 1340-1353.
- Stowers, J., Hayes, M., & Bainbridge-Smith, A. (2011). *Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor*. 2011 IEEE International Conference on Mechatronics, 358–362.

- Team, C. (2019). Install Ubuntu Server | Ubuntu tutorials. [online] Tutorials.ubuntu.com. Available at: <https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-server#0> [Accessed 20 Sep. 2019].
- Tech Explorist. (2019). Baxter Robot: The Blue-Collar Robot. [online] Available at: <https://www.techexplorist.com/baxter-robot-blue-collar-robot/2452/> [Accessed 20 Sep. 2019].
- Techterms.com. (2019). SDK (Software Development Kit) Definition. [online] Available at: <https://techterms.com/definition/sdk> [Accessed 20 Sep. 2019].
- The Economic Times. (2019). New robotic drones can travel by land and air. [online] Available at: <https://economictimes.indiatimes.com/news/science/new-robotic-drones-can-travel-by-land-and-air/articleshow/59353384.cms?from=mdr> [Accessed 20 Sep. 2019].
- The Interaction Design Foundation. (2019). Natural User Interfaces – What are they and how do you design user interfaces that feel natural?. [online] Available at: <https://www.interaction-design.org/literature/article/natural-user-interfaces-what-are-they-and-how-do-you-design-user-interfaces-that-feel-natural> [Accessed 20 Sep. 2019].
- Tholey, G.; Desai, J.P.; , "A General-Purpose 7 DOF Haptic Device: Applications Toward Robot-Assisted Surgery," *Mechatronics*, IEEE/ASME Transactions on , vol.12, no.6, pp.662-669, Dec. 2007
- Ugr.es. (2019). Pick and Place. [online] Available at: https://www.ugr.es/~cvrlab/projects/baxter_objmanipulation/baxter.html [Accessed 20 Sep. 2019].
- University, S. (2019). Stanford professors discuss ethics involving driverless cars | Stanford News. [online] Stanford News. Available at: <https://news.stanford.edu/2017/05/22/stanford-scholars-researchers-discuss-key-ethical-questions-self-driving-cars-present/> [Accessed 20 Sep. 2019].
- University, S. (2019). Stanford's humanoid robotic diver recovers treasures from King Louis XIV's wrecked flagship | Stanford News. [online] Stanford News. Available at: <https://news.stanford.edu/2016/04/27/robotic-diver-recovers-treasures/> [Accessed 20 Sep. 2019].
- Voosen, K. (2011) 3D Vision and the Kinect. Retrieved from ftp://ftp.ni.com/pub/branches/uk/nidays2011/Designing_Vision_Systems_including_3D_Vision_and_the_Microsoft_Kinect.pdf (Accessed 4th September 2019)
- Weber, C., Hahmann, S., & Hagen, H. (2010a). *Sharp Feature Detection in Point Clouds*. 2010 Shape Modeling International Conference, 175–186.
- Wiki.ros.org. (2019). ROS/Concepts - ROS Wiki. [online] Available at: <http://wiki.ros.org/ROS/Concepts> [Accessed 20 Sep. 2019].

