Teleoperation Control of Baxter Robot using Body Motion Tracking

H. Reddivari[1], C. Yang*[1,2], Z. Ju[1,3], P. Liang[1,4], Z. Li[2] and B. Xu[5]

***Abstract*— In this paper, we use Kinect Xbox 360 sensor to implement the motion control of Baxter robot, a semi humanoid robot with limbs of 7 DOF joints with collision avoidance capabilities. Two different methods using vector approach and inverse kinematics approach have been designed to achieve a given task. Primitive experiments have been carried out to verify the effectiveness of the developed methods. Human motions is captured by Kinect sensor and calculated with Processing Software using SimpleOpenNI wrapper for OpenNI and NITE. UDP protocol is adopted to send reference motion to Baxter robot. Python and RosPy script programming kit is used to calculate joint angles of Baxter robot based on vector approach and inverse kinematics approach. The experimental results demonstrate that both of our proposed approaches have achieved satisfactory performance.**

I. INTRODUCTION

With rapid development of automation technology, the application of robots in industries has been improved largely. Robots are used in industries for various applications, e.g. operation of some replicated or dangerous tasks instead of human . The robot is considered to be able to adapt to the industrial changeable needs, however, most robots currently can only be operated by kinds of devices such as keyboard, joysticks, or tactile devices which may be complicated due to special complicated tasks. Consequently, the robot need to be operated and instructed by professional staffs or people need training which may increase the cost and extra time. Human have the ability to adapt to the environmental disturbances, so it would be efficient if robot is operated according to the human skills in real time. A straightforward way to achieve this goal is to let robot follow the human motion and body tracking can be an ideal method to transfer human skills to robot side.

To implement human body tracking, human body itself should be tracked first. Presently there are a number of approaches to human body tracking. The most approach is to fix tracking markers to human body but this may cause a lot of inconvenience to user. Another approach is image processing using normal cameras. This method is not reliable because of the poor body detection capabilities of image processing. Depth analysis using stereo-vision cameras is also applied in body tracking which needs long processing time and may not be effective in real time. In this paper,Kinect Xbox 360 is used to track human body motion. Its depth sensor is accurate and can be used in real-time effectively.

As Kinect sensor gives the data in Cartesian space which has to be converted into joint space in order to interact with the robot. In this paper, two main methods are proposed to solve this problem, that is the vector approach and inverse kinematics approach and a comparison between the two methods is also discussed. This paper will first give a brief of the preliminary knowledge, then introduce the setup of the robot and finally the results are compared.

II. TELEOPERATION SYSTEM

A. System configuration

To illustrate the tele-operation of robot using body tracking, a demonstrative syetem was built. It consisted of the body tracking device and a robot, see Fig. 1.

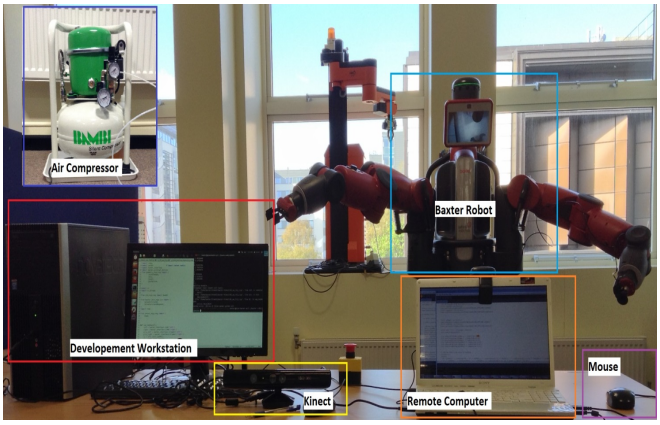

Fig. 1. Enviroment of teleoperation control with kinect

Body tracking is done with *Kinect Xbox 360* sensor. Although there are different types of Kinect, *Kinect Xbox 360* is used because its price is low and it can satisfy all the requirements of the project. The Kinect device is connected to *Remote Computer*. On this computer, *Processing* software was installed and used to receive the position data from Kinect sensor.

The robot used here is Baxter Research robot by Rethink Robotics. It is a semi humanoid robot with limbs of 7 DOF joints and has collision avoidance capabilities. It can be controlled by torque, velocity and position mode. The robot was connected to and controlled by a computer, *Development*

* Corresponding author: cyang@ieee.org
[1]Center for Robotics and Neural Systems, Plymouth University, Devon PL4 8AA, UK; [2] Key Laboratory of Autonomous System and Network Control, College of Automation Science and Engineering, South China University of Technology, Guangzhou, 510640, P. R. China; [3]North Automatic Control Technology Institute, Taiyuan, 030006, P. R. China; [4]The State Key Laboratory of Robotics and System, Harbin Institute of Technology, 150080, P. R. China; [5]School of Automation, Northwestern Polytechnical University, 710072, P. R. China;

This work was supported in part by EPSRC grants EP/L026856/1 and EP/J004561/1; Royal Society grants IE130681 and RG130244; and National Natural Science Foundation of China under Grant 61473120.

Workstation, it is connected to *Remote Computer* with Ethernet cable. A accessory device, *Air Compressor*, was used to provide vacuum that make the robot grip object.

B. System principle

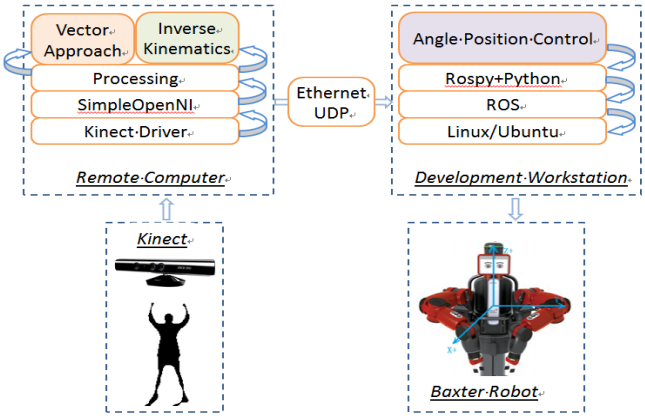

Fig. 2. Principle of teleoperation control with kinect

C. Kinect Sensor

Kinect Xbox 360 used in the proposed teleoperation system is a set of sensors developed as a peripheral device with Xbox game device. It is developed by Microsoft and widely used in human motion 3D tracking such as body language, gestures etc. via its depth camera [1]. There is a RGB camera and a dual infrared depth sensor in *Kinect*. Using image and depth sensors, *Kinect* can detect movements of a user. It does not need user to wear any kind of accessory. The image and depth sensors are housed in a base with a motor that allows it to adjust sensors up and down, see Fig.3. Two sensors make

Fig. 3. Kinect Sensors. 1. Depth sensors, 2. RGB camera, 3. Motorized base

up the depth component of the Kinect: a monochrome CMOS sensor and an infrared projector(label 1, Fig.3). Together, these are the basis for body tracking.

Fig. 4 shows RGB and depth image of Kinect device. Human body can be simplified by chopsticks representing position and posture of human in 3D space. In teleoperation system, Kinect maps human joint positions and velocities to robot side, which enable human-robot interaction smoothly in real time. Compared with conventional motion tracking system based on RGB data which requires complicated programming, high expense and worn by users, Kinect can be embedded in the control system via open source drivers

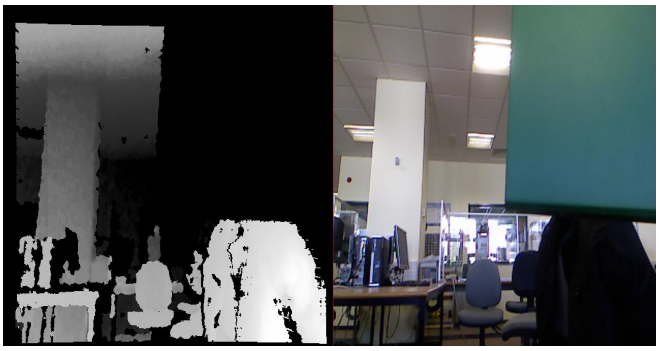

Fig. 4. Image showing depth and RGB sensor data of Kinect

e.g. SDK, openNI. Furthermore, its low cost and friendly human machine interface make teleoperation tasks naturally and robust to environmental changes.

D. Kinect Develop Software

Many softwares are available to interface Kinect with PC, e.g., Libfreenect by OpenKinect, OpenNI, Microsoft Kinect for windows SDK [3], [4].

OpenKinect is an open community for people interested in making use of the Kinect hardware with PCs and other devices. It releases a software called Libfreenect which can be used to interface Kinect along with any operating system in PC [3], [5].

OpenNI (open natural interaction) is a software like Libfreenect[4]. It can support many other devices other than Kinect. It is an open source software released by Prime Sense, OpenNI uses middle ware called NITE to get the skeleton data of the user [6]. Microsoft Kinect for windows SDK is another similar platform launched by Microsoft for windows systems[7]. SimpleOpenNI that uses OpenNI and NITE, [6] is an OpenNI and NITE wrapper for processing. It may not support all the functions that are supported by OpenNI and NITE still, but most of the useful features can be accessed in a simple way [2], [8].

The choice of software is very important and should be on the basis that: **(i)** capability of extracting skeletal data; **(ii)** compatibility with multiple platforms such as Windows and Linux; **(iii)** good documentation; and **(iv)** simplicity for fast verification of algorithms. After properly comparison, *Processing* software which satisfies all the requirements is employed. It can be interfaced with Kinect with SimpleOpenNI wrapper for OpenNI and NITE [8], skeleton data can be extracted and used, and support both Windows and Linux platform[13], it has a good documentation about the library, it's simple, fast and effective to verify the test algorithms. *Processing* is built on Java, so its functionality is very similar to Java. The useful functions of Processing that are employed in this work are detailed as below:

PVector: A class to describe as two or three dimensional vectors, specifically a Euclidean (also known as geometric) vector. A vector is an entity that has both magnitude and direction. The data type, however, stores the components of

the vector (x, y for 2D, and x, y, z for 3D). The magnitude and direction can be accessed via the methods mag () and heading (). See [9] for more details.

pushMatrix() and popMatrix(): Push the current transformation matrix onto the matrix stack. The pushMatrix() function saves the current coordinate system to the stack and popMatrix() restores the prior coordinate system. The pushMatrix() function and popMatrix() function are used in conjunction with the other transformation functions and may be embedded to control the scope of the transformations [10].

E. ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.[14]

F. Rospy

Rospy is a pure Python client library for ROS. The rospy client API enables Python programmers to quickly interface with ROS Topics, Services, and Parameters. The design of rospy favors implementation speed (i.e. developer time) over runtime performance so that algorithms can be quickly prototyped and tested within ROS. It is also ideal for non-critical-path code, such as configuration and initialization code. Many of the ROS tools are written in rospy to take advantage of the type introspection capabilities. Many of the ROS tools, such as *rostopic* and *rosservice*, are built on top of rospy [14], [15], [16].

G. UDP protocol

The User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite (the set of network protocols used for the Internet). With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths.UDP is suitable for purposes where error checking and correction are either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.[17]

H. Baxter research robot

The experiment platform used in this paper is the baxter robot, which consists of a torso, 2 DOF head and two 7 DOF arms (shoulder joint: s0, s1; elbow joint: e0, e1; wrist joint: w0, w1, w2), integrated cameras, sonar, torque sensors, encoders and direct programming access via a standard ROS interface etc. Each joint of the baxter robot arm is driven by 7 SEAs (Serial Elastic Actuators), which provides passive compliance to minimize the force of any contact or impact. ROS (Robot Operating System) SDK is used to control and program the baxter robot with baxter RSDK running on Ubuntu 12.04 LTS. ROS is an open source framework with module tools, libraries, and communications. It simplifies the task of modeling and programming on a wide variety of robotic platforms. The setup of the Baxter robot in this work is depicted in Fig. 1.

III. Design Principles

A. General calculation

Most of movement calculations are based on two or more locations, distances and joint angles to calculate the distance between two points for 2 and 3 dimensional points, refer to equations 1 and 2.

$$d_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

$$d_3 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2}$$

where (x_1, y_1) and (x_2, y_2) are points in 2D space, d_2 is the distance between these two points , (x_1, y_1, z_1) and (x_2, y_2, z_2) are points in 3D space, d_3 is the distance between these two points. The law of Cosines can help to calculate the angle between the joints. The maximum computable angle is 180 degrees. When calculating the angle between the joints, an additional point is needed to determine 180∼360 degrees. From the skeleton tracking data, we can draw a triangle using any two joint points. The third point of the triangle is derived from the other two points. If the coordinates of each point of the triangle is known, we can know the length of each side, but can not know the angle values. As shown in Fig.5, by applying the law of Cosines, the magnitude of any desired angle can be calculated.

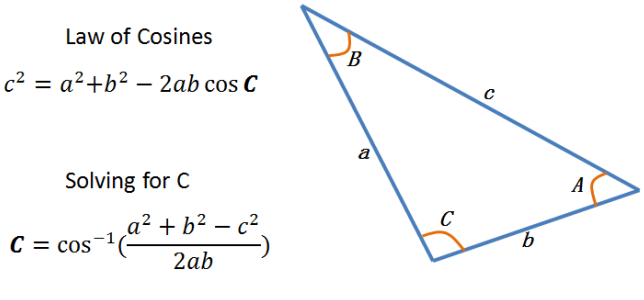

Fig. 5. Law of Cosines

Calculations on the joint points give the length of sides a, b, c. The angle of triangle can be calculated by the law of cosines.

B. Vector Approach:

Kinect can detect all the coordinates of the joints of human body and return its location coordinates. These coordinates can be converted into vectors and the respective angles of the joints can be calculated. In this method the Cartesian coordinates of the body joints are extracted from Kinect and the respective angles from limbs are calculated. They are then sent to Python code that interact with Baxter, after mapping them according to our requirement. First, the four angles

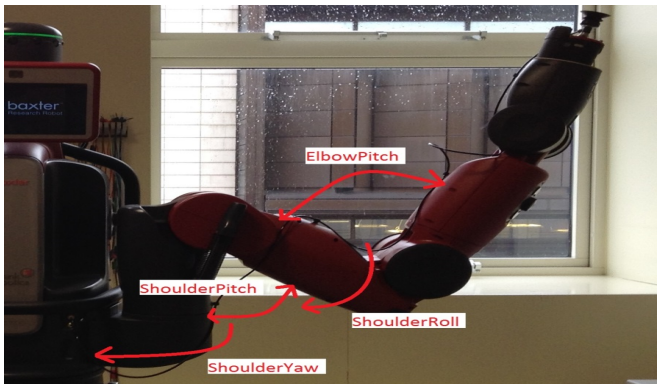

Fig. 6. Angles used in Vector Approach: Shoulder pitch, Shoulder Yaw, Shoulder Roll and Elbow Pitch.

including Shoulder Pitch , Shoulder Yaw, Shoulder Roll and Elbow Pitch, as shown in Fig. 6, are calculated from the limb position coordinates that are obtained from the Kinect.

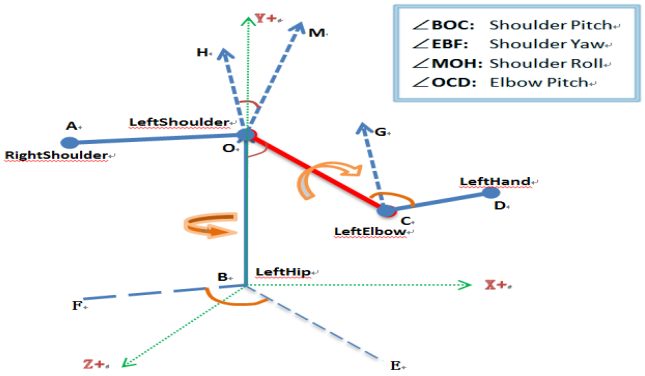

Fig. 7. The principle of angles calculation in vector approach.

The principle of angles calculation using vector approach is shown in the Fig. 7. The bold lines CO and CD represent left upper arm and lower arm of human respectively. Bold line BO is a line from left hip to left shoulder, and AO is a line from right shoulder to left shoulder. The directed segments BX+, BY+ and BZ+ represent the axis of frame in Cartesian space of kinect, and the point B is the origin of the frame.

Calculation of Shoulder Pitch and Elbow Pitch: As shown in Fig. 7, the shoulder pitch angle($\angle BOC$) is calculated from the angle between two vectors $\overline{OB}$ and $\overline{OC}$. The calculation can be solved by using the positions of three joints, namely, hip(point B), shoulder(point O) and elbow(point C).Pass the three points to the angleOf() function, This function returns the angle that can be directly sent to Baxter. Elbow pitch ($\angle OCD$, the angle between $\overline{OC}$ and $\overline{CD}$) can be calculated by passing hand, elbow and shoulder points into the angleOf() function as well. In fact, any angle between two vectors in the Processing Software can be calculate by using the angleOf() function via this method.

Calculation of Shoulder Yaw: As shown in Fig. 7, the shoulder yaw angle($\angle EBF$) is calculated in a similar way by using both shoulder point(point A, O) and elbow point(point C), making up of the vectors $\overline{OC}$ and $\overline{OA}$. But here these two vectors($\overline{OC}$ and $\overline{OA}$) are projected into the XZ plane to get the vectors $\overline{BE}$ and $\overline{BF}$. Then shoulder yaw ($\angle EBF$, the angle between $\overline{BE}$ and $\overline{BF}$) is calculated by using angleOf() function.

Calculation of Shoulder Roll: Among the angle calculations, shoulder roll is the most challenging task. As the calculation is not intuitive and all the obtained points are in 3D plane, the same method used for the previous angle calculations cannot be available.

The idea here is to find the angle made by the elbow-hand vector in plane perpendicular to the shoulder-elbow vector and passing through the shoulder joint. The reference vector must be stable with respect to the body. So this reference vector is calculated by taking cross product between shoulder-shoulder vector and shoulder-elbow vector.

In this case, the *normal line* from crossproduct of two vectors can be used. First, the vector $\overline{OM}$ can be gotten by calculating the crossproduct of vectors $\overline{OC}$ and $\overline{OA}$. The vector $\overline{OM}$ is perpendicular to plane decided by the vector $\overline{OC}$ and $\overline{OA}$, is the *normal line* vector of this plane. Obviously, vector $\overline{OM}$ is perpendicular to vector $\overline{OC}$(left upper arm). In this way, the *normal line* vector $\overline{CG}$ can be calculated from cross product of vector $\overline{OC}$ and $\overline{CD}$, which is also perpendicular to vector $\overline{OC}$. Then, translating vector $\overline{CG}$ along the vector $\overline{CO}$ to point O can get a vector $\overline{OH}$. The angle between vectors $\overline{OH}$ and $\overline{OM}$, $\angle MOH$, is the shoulder roll angle.

The orientation data sent by the Kinect can be extracted by using PMatrix3D instance in Processing software. The orientation matrix is given into the PMatrix3D variable. Then, the present coordinate system is pushed and saved into the stack. The coordinate system is then moved to the shoulder joint and the orientation matrix is used to orient the transformed coordinate axis. All the calculations in this function from here will be calculated in this transformed coordinate system.

After the calculation of roll angle, the original coordinate system is regained by popping the matrix from the stack. The right shoulder roll is also calculated in the similar way. A small change has to be made in the direction of the vectors.

A little error correction is made to the shoulder roll, because the function used to find the roll is not completely accurate. The shoulder roll is observed to be changing with the shoulder yaw. So when the data of shoulder roll and shoulder yaw are sent to the MATLAB and plotted the relationship observed in Fig. 8. From trial and error method, the error is partially corrected by the equation shown below:

$$leftShoulderRoll = \\ -leftShoulderRoll - leftShoulderYaw/2 - 0.6 \quad (3)$$

Now these returned angles are sent to the Python code in the Baxter development work station using UDP communication. The function shown above will send the data packets through the server object created earlier. Now that all the angles are

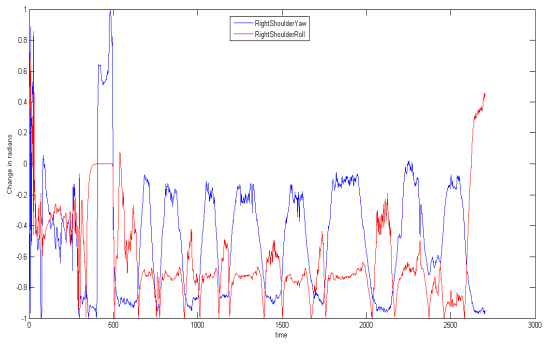

Fig. 8. Error of Vector Approach

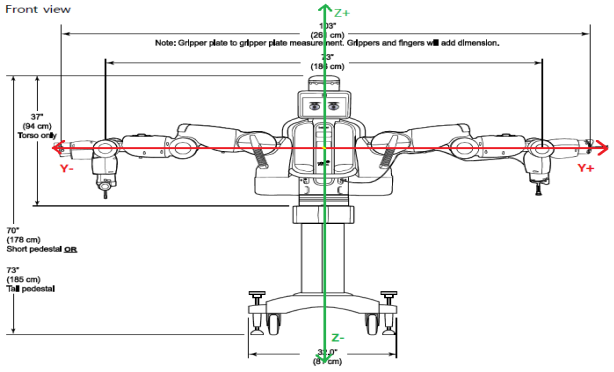

Fig. 9. Coordinate map between human hand and Baxter robot: Front view of robot

sent to the Python code, and Python code only has to use these angles to control Baxter robot.

C. Inverse Kinematics

Inverse kinematics is referred to calculation of joint angles from the end effector coordinate location using kinematic equations and the constraints such as length and angle constraints of the robot. Kinect gives the coordinate location of the hand. These coordinate locations can be converted to the required joint angles of the Baxter robot.

Coordinates Extracted: Firstly the coordinates of the hand joints are extracted used for controlling the end effector by inverse kinematics, then the elbow coordinates are extracted. These coordinates together with hand coordinates can be used to calculate the length of the joints.

Mapping human hands with Baxter hands: Human hands are not of the same size of those of Baxter robot. So it is essential to map human hands with Baxter hands for the inverse kinematics method to work correctly. As mentioned earlier the length of the hands can be found by calculating the magnitude of the hand-elbow vector. The Baxter limb joint is found to be 0.47 in its coordinate system.

Analyzing Baxter coordinate system: The Baxter coordinate system convention(See Fig. 9, Fig. 10) is not as same as Kinect coordinate system convention(See Fig. 11). So the coordinate axis must be mapped according to the respective conventions.

$$\begin{aligned} X_{Baxter} &= -Z_{Kinect} \\ Y_{Baxter} &= X_{Kinect} \\ Z_{Baxter} &= Y_{Kinect} \end{aligned} \tag{4}$$

After mapping, the coordinates can be sent to the Python code to control Baxter. An error value of 2.2 is added to the Z coordinate in order to compensate for the error.

IV. Experiment and Demostration

The results of the experiments [1] are described as below.

[1] An illustrative video of the experiment can be viewed at: https://www.youtube.com/watch?v=jHjvEHT47dk or http://v.youku.com/v_show/id_XNzU1NTc2ODc2.html

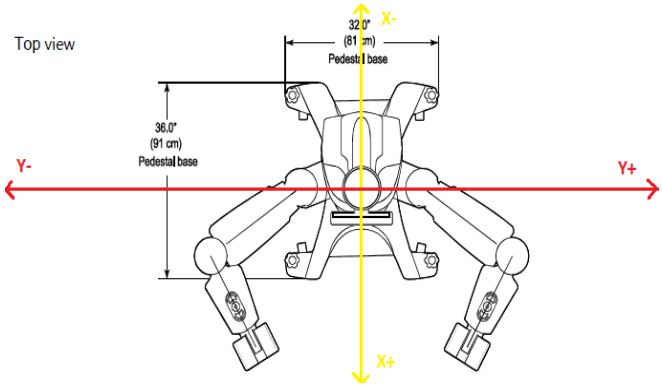

Fig. 10. Coordinate map between human hand and Baxter robot: Top view of robot

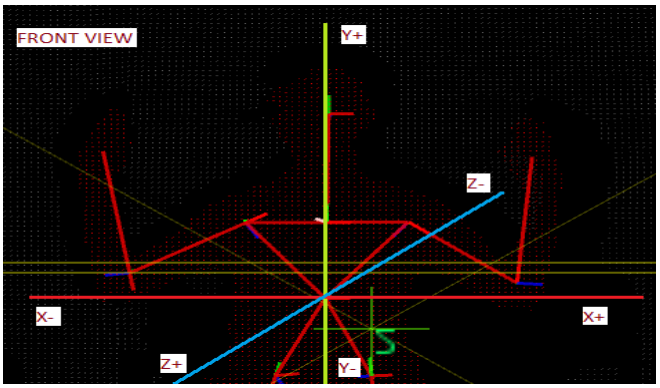

Fig. 11. Coordinate map between human hand and Baxter robot: skeleton of human

A. Results of vector approach method

When every step is done, a small error is observed in the received angles and the required angles. So the received angles and the required angles are mapped with the map command in Processing software. It is observed that when the required angle is -0.6, the received angle is -0.1 and when it is 0.6, the received angle is -0.9. So these values are mapped accordingly. Similarly, all the angles are corrected according to the required angles. In the end the Baxter works as expected, but by using this method only 4 of 7 joints are

controlled. In order to control the remaining joints extra data of the palm and fingers is required, which is not available from the Kinect. This problem can be solved in the inverse kinematics approach.

B. Results of Inverse Kinematics method

Processing is able to extract the coordinates from the Kinect and mouse click events from the mouse. The network protocol is able to send the data and the development workstation successfully receives the values sent by the Processing. The Python code cannot solve all the positions sent by the Processing. Baxter responds to all the solved positions at the same time avoiding collision between hands. The grippers and suction are able to respond to the mouse clicks. A new user may find it little difficult as the forward and backward movement of robot limbs is mirror image to the human hand motion, whereas the sidewards motion of the limbs will follow the human hands.

V. CONCLUSION

In this paper, Kinect sensor is used to control robots with both vector approach and inverse kinematics approach. Both the methods of vector approach and inverse kinematics have its own advantages and disadvantages. The Vector approach can exactly follow human arm, but only 4 of the 7 joints can be controlled. The inverse kinematics method can be used in picking and placing industrial applications, but cannot exactly replicate the human motion. Alongside advantages of the proposed control methods, there are some disadvantages as well. The PSI pose has to be made by every user in order for the Kinect to detect the user perfectly. Due to some technical reasons, the program does not support too tall or too short people. Some of the data that is sent by the Processing software to development work station is faulty and needs to be filtered. Filtered data may not have a joint solution in the Inverse Kinematics approach. So sometimes lag/delay is observed in the robot motion.

REFERENCES

[1] E. Machida, Meifen Cao, T. Murao, and H. Hashimoto. Human motion tracking of mobile robot with kinect 3d sensor. In *SICE Annual Conference (SICE), 2012 Proceedings of*, pages 2207–2211, Aug 2012.

[2] Borenstein, G. Making Things See: 3D vision with Kinect, Processing, Arduino, and MakerBot "O'Reilly Media, Inc.", 2012

[3] Fabian, J.; Young, T.; Peyton Jones, J.C.; Clayton, G.M., "Integrating the Microsoft Kinect With Simulink: Real-Time Object Tracking Example," Mechatronics, IEEE/ASME Transactions on , vol.19, no.1, pp.249,257, Feb. 2014

[4] Colvin, C.E.; Babcock, J.H.; Forrest, J.H.; Stuart, C.M.; Tonnemacher, M.J.; Wen-Shin Wang, "ultiple user motion capture and systems engineering," Systems and Information Engineering Design Symposium (SIEDS), 2011 IEEE , vol., no., pp.137,140, 29-29 April 2011

[5] $http://www.openkinect.org/wiki/Main_Page$

[6] Chye, C.; Nakajima, T., "Game Based Approach to Learn Martial Arts for Beginners," Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on , vol., no., pp.482,485, 19-22 Aug. 2012

[7] Soltani, F.; Eskandari, F.; Golestan, S., "Developing a Gesture-Based Game for Deaf/Mute People Using Microsoft Kinect," Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on , vol., no., pp.491,495, 4-6 July 2012

[8] Cruz, L.; Lucio, D.; Velho, L., "Kinect and RGBD Images: Challenges and Applications," Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on , vol., no., pp.36,49, 22-25 Aug. 2012

[9] $http://processing.org/reference/PVector.html$

[10] $http://www.processing.orgreferencepushMatrix_html$

[11] Fitzgerald, C., "Developing baxter," Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on , vol., no., pp.1,6, 22-23 April 2013

[12] $https://github.com/RethinkRobotics/sdk-docs/wiki/Baxter-Overview$

[13] $https://code.google.com/p/simple-openni/$

[14] $http://en.wikipedia.org/wiki/Robot_Operating_System$

[15] Zaman, S.; Slany, W.; Steinbauer, G., "ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues," Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International , vol., no., pp.1,5, 24-26 April 2011

[16] $http://wiki.ros.org/rospy$

[17] $http://en.wikipedia.org/wiki/User_Datagram_Protocol$