Proceedings of the 2017 International Conference on Machine Learning and Cybernetics, Ningbo, China, 9-12 July, 2017

DEVELOPMENT OF A MIXED REALITY BASED INTERFACE FOR HUMAN ROBOT INTERACIOTN

MATTHEW COUSINS[1],CHENGUANG YANG[1],JUNSHEN CHEN[1],WEI HE[2],ZHAOJIE JU[3]

[1]Zienkiewicz Centre for Computational Engineering, Swansea University, SA1 8EN, UK
[2]School of Automation and Electrical Engineering, University of Science and Technology Beijing, Beijing 100083, China
[3]School of Computing, University of Portsmouth, PO1 2UP, UK
E-MAIL: cyang@theiet.org

Abstract:

This research paper presents a novel interface of programming by demonstration (PbD) for Human-Robot Interaction (HRI). The designed mixed reality (MR) based interface is providing users a more immersive user experience (UX) while teleoperating the robot. The operator's hand gestures are captured and be used to control the robot. Users are able to see their own hands in the virtual environment. Experimental test was carried out with a dual-arm robot to make the robot react to a gesture made by an operator viewing the robot workspace through a virtual reality (VR) headset.

1 Introduction

The aim of this paper was to use the emerging technology of MR to improve interaction with robot. There are several methods for programming robots to complete tasks. Coding to instruct the robot is a popular technique, but teaching by direction is also a common method of programming [13]. This method involves physically moving the robot by hand to show it the routine it must follow to complete the task [1]. Robots, as with any other machinery, may need adjustment while in use. Both tasks require direct human interaction, which can be dangerous due to the crushing power of the robots. It is important to consider health and safety risks; injuries have occurred from robots malfunctioning while people have been in close proximity to them [9]. Robots are also often used in hazardous environments where human presence is not possible. This means that the robots cannot be moved or programmed as the environment they are in is hazardous to humans which does not allow direct contact between human and robot. Examples of these hazardous environments could include nuclear sites when dealing with the movement and storage of fuel rods and radioactive materials, or in other cases such as handling chemicals that are harmful to human health. This results in the inability to teach the robot by direction, or correct/adjust the robot in the event of a problem.

VR is a technology that presents digitally created information to our senses via various pieces of equipment, that in combination gives the user the feeling that they are in the virtually created environment. This can be achieved using many different pieces of hardware to cover each of the human senses, e.g. headsets, omni-directional treadmills and special gloves. In [2], by wearing a visual headset, operators can intuitively control the pose of a camera on the head of the robot, and the operator is able to perceive from the robot's perspective. Combining these pieces of technology simulates senses in unison to create the feeling of reality. A VR-based interface was created for Underwater Robots that utilizes immersive technologies to reduce user faults and mental fatigue [4].

Augmented Reality (AR) is 'technology to superimpose information on the world we see'. This technology is used to take normal surroundings and add to what is already there to create different experiences or enhance day-to-day life. There have been various attempts over the past few years to bring this technology to main stream consumers. For example, a novel AR system was developed for defining virtual obstacles, specifying tool positions, and specifying robot tasks [5]. An augmented reality visualization interface was presented in [7] to simultaneously present visual and laser sensors information further enhanced by stereoscopic viewing and 3-D graphics.

There have also been applications in entertainment with devices such as the Nintendo 3DS using AR to create games that interacted with the player's surroundings. AR is different from VR; instead of using entirely computer-generated environments to fool the user into thinking they are somewhere else, and immerses them in a new environment, AR just adds to the environment they are already in. Programming by demonstration (PbD) is a similar technique to the record and replay technique, in which a robot is shown a set of movements and then repeats them exactly multiple times, but PbD has an aspect of learning integrated into it, making a more effective system. The main aim of programming by demonstration is that the end user of the robot can teach the robot to perform a task without the need for programming [1]. Before this technology became wide spread every function a robot would undertake needed to be meticulously broken down and analysed so a programmer would be able to code in each step individually. In this traditional case, a programmer would have been able to programme the robot to be able to react to every possible case within its working environment. To do this effectively the task has to be broken down into numerous steps and each step tested thoroughly with many different scenarios to ensure the robot is able to cope with these changes [1] [12].

In contrast to conventional coding procedure, programming by demonstration allows this process to be streamlined by showing the robot its task, while its position, joint rotations and any other required pieces of data are recorded. This allows it to repeat the task by following this data and no coding is required. It is also possible for the robot to learn how to deal with varying circumstances by showing it through multiple different but similar scenarios and the robot will be able to generalise its task [1] [12]. Combining cutting edge technology, VR, with PbD is an effective way of teaching robot. In [10], the author created a cleaning robot with virtual agents in a virtual environment. The proposed VR based PbD method can help to reduce the labor, time and cost associated with interactive learning of robot. An VR approach provides an effective method of learning by imitation and programming by demonstration was also presented in [6].

This paper aimed to allow people to interact with robots from a remote location where they cannot be harmed, interact with robots in hazardous environments and enable one technician to interact with as well as oversee multiple robots from a single location. VR was the tool that allowed this to happen. Using a webcam, the robot and its surroundings were recreated in a VR headset. A VR headset is a device that a user wears on their head, which through a compatible video input, can recreate scenarios with the correct field of view and depth perception in 3D. Wearing the headset and using the camera as the video input created a realistic perception of the cameras location, however as a webcam was used the image was 2D. A LEAP Motion Controller was then used in combination with the VR headset device. The motion controller is a device that is able to track a user's hand movements and recreate a virtual model of the hands to be used for interacting with the robot.

Using the combination of this hardware and complimentary software, the user was able to get a realistic view of the robot and its surroundings, and in combination with the LEAP Motion would be able to see their hands overlaid on this view. This enabled the user to see and interact with the robot from a remote location by 'touching it' with virtual hands or using hand gestures. Also, the depth provided by the VR headset would allow for accurate adjustment of the robot in a 3D space. The hardware layout diagram in Fig. 1 shows in more detail how the hardware was laid out and how it interacted with each other to achieve this goal.

The features mentioned above provide an accurate interaction from a safe and remote position. Using this technology, one operator is able to interact with multiple robots from one location. This means adjustments can be made quickly as it would not require the operator to go to the specific location of the robot.

In this paper, the method used to achieve the MR human robot interaction will be laid out and explained, showing steps taken to overcome various problems and what steps were needed to re-create the result. The purpose of this paper was to allow for MR human robot interaction; therefore, this paper will demonstrate that it is possible and that live control of the Baxter robot can be achieved. The controller was used to track the hands of the user and obtain data that will be used by software to adjust the Baxter robot. It was also used to render virtual hand models of the user's hands to be overlaid on a webcam view of the robot, shown through the VR headset.

2 System Description

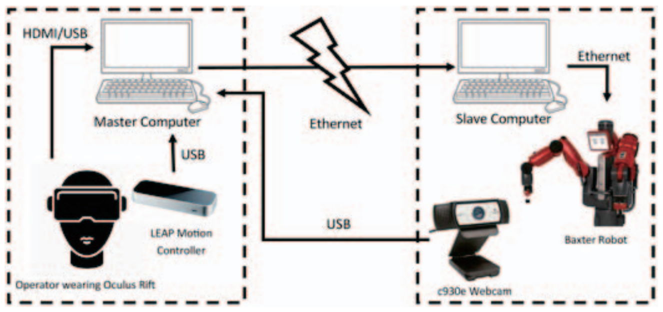


FIGURE 1. The illustration of the teleoperation system.

2.1 LEAP Motion Controller

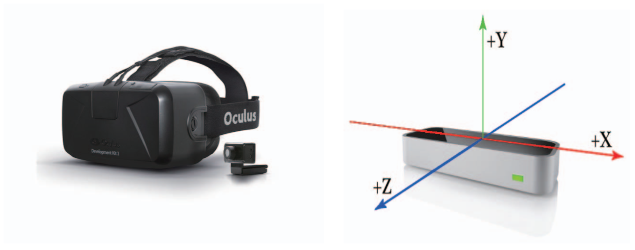
The LEAP Motion Controller is a suite of hardware and software designed for untethered VR/AR headsets. It is a tracking device that aims to replace the use of controllers in VR applications such as the HTC Vive Controllers and Oculus Touch. It does this by tracking the user's hands in a 180-degree ark in front on the user, up to distances slightly greater than arm's length. By tracking individual joints in the hand and finger tips, the LEAP Motion is able to produce an articulated computer generated model of the user's hands. These models can then be used in VR applications and as the controllers in the VR environment. This can help to improve immersion in the VR environment as it creates a more specific and realistic interaction as the use of hands is closer to the real world. LEAP Motion technology also has productivity applications. A novel markerless human-robot interface was created by using LEAP motion. The manipulator copies the movements of human hands [3]. A system for control robot arm using LEAP Motion Controller was designed and tested in [8]. LEAP Motion can also be used in a desktop mode where gestures for Windows can be enabled; meaning gestures above the keyboard can be pre-programmed for set tasks such as a 'swipe down' to close all windows. A LEAP Motion model with its coordinates system was shown in Fig. 2(b).

2.2 Oculus Rift Development Kit 2

The Oculus Rift DK2, as shown in Fig. 2(a), is a VR headset from Oculus that was produced so developers could work on VR applications while the consumer edition headset was still in development.

A user wearing this headset can see an image displayed on a low latency high definition screen in front of their eyes. This image is then morphed through a set of lenses so that the user sees a high definition 3D image, to make them feel like they are really in the 3D environment. The headset, with the help of a sensor mounted on the monitor, is also capable of tracking the position of the user's head in 3D space, as well as the head angle of the user. This allows the head position and angle to be replicated in the virtual environment, again adding to the immersion of the environment making the user feel like they are really there.

This iteration of the headset also came with several engine integrations, allowing for much better interaction with common game engines such as Unreal Engine 4 and CryEngine. It also includes the Oculus SDK, which contains source code for interacting with and using the device via code.



(a) Oculus Rift DK2 (b) LEAP Motion

FIGURE 2. Oculus Rift DK2 and LEAP Motion.

2.3 Baxter® robot arm

The Baxter robot is a collaborative robot from Rethink Robotics. It is a robot designed for use in 'low volume, high mix production jobs' such as line loading, material handling and packaging. Baxter can also be trained by demonstration instead of being programmed, allowing it to be trained by existing personnel for a certain job. Baxter's arms have 7 Degrees of Freedom (DOF), as shown in Fig. 3, which enable them to move like human arms, this allows the robot to complete more tasks and use less space.

Inverse Kinematics allows a tasks space trajectory to be inputted and will find a viable joint trajectory that will replicate the input trajectory. The differential kinematics equation, in terms of either the geometric or the analytical Jacobian, establishes a linear mapping between joint space velocities and task space velocities [11].

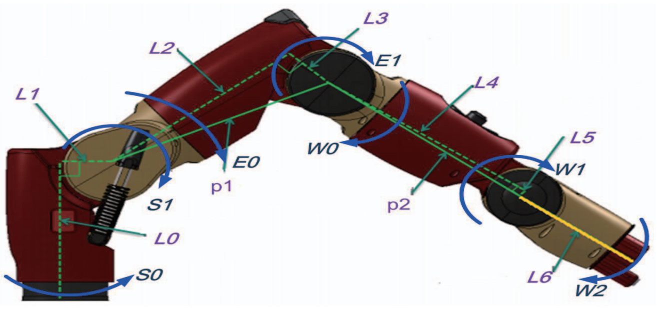


FIGURE 3. Kinematic model of Baxter arm.

3 Methods

The first task to be undertaken was to implement each individual aspect of the project. The LEAP Motion hand tracking and the camera pass through, separately before attempting to combine them together.

3.1 LEAP Motion Hand Tracking

The first component to be created was the hand tracking of the LEAP Motion in Unreal Engine. The LEAP Motion plug-in guide was used to find what was needed to bring the hands into the virtual environment. By creating a new Blueprint, LeapGameMode, which would set the game mode of the level to use a LEAP Motion Character as its default pawn. A pawn in Unreal Engine is a character that can be interacted with within the level, and this Blueprint sets this character to be one related to the LEAP Motion Controller. In this case, the pawn used was the LeapFloatingHandsCharacter. This was a simple and easy to manipulate character, which used only floating hand models. There were other pawn options provided by the plug-in however these added features were not needed, such as a whole body model, which could over complicate things further on in the project so were avoided in favour of the simpler pawn. The Heads-Up Display (HUD) class was also set to enable a heads-up display in this game mode.

3.2 Unreal4AR AR Plug-in

In order to get the web camera feed from the outside world into the development environment, the Unreal4AR Plug-in was used. Running the example programme for this plug-in opens a window showing the webcam feed. This shows that the editor is receiving the webcam footage, however this footage is not being displayed in the headset.

By using the VR Preview option, Unreal Editor automatically converted the normal windowed output for use in the VR headset. The two different outputs are shown in Fig. 4.

The view was now in the correct format, however the view in the headset was very far away. The webcam view needed to fill the view of the user to be immersive and to look like reality.

By editing the base component ARToolkitBase, which is the item in the level that is responsible for the webcam feed, the webcam view was adjusted to fill the view of the user to be immersive and to look like reality. The larger component is made up of two constituent parts; the camera, which is the users view port, and the screen. The screen is a large square shape in the level, that has a special active texture assigned to its surface. Through the plug-in, the active texture displays the image from the webcam.

To increase the size of the image, the screen needed to be moved closer to the camera, while keeping the cameras field of view the same. The distance to the screen worked just as it would in real life and as the screen was moved closer to the user, the image appeared bigger until the users field of view was filled.

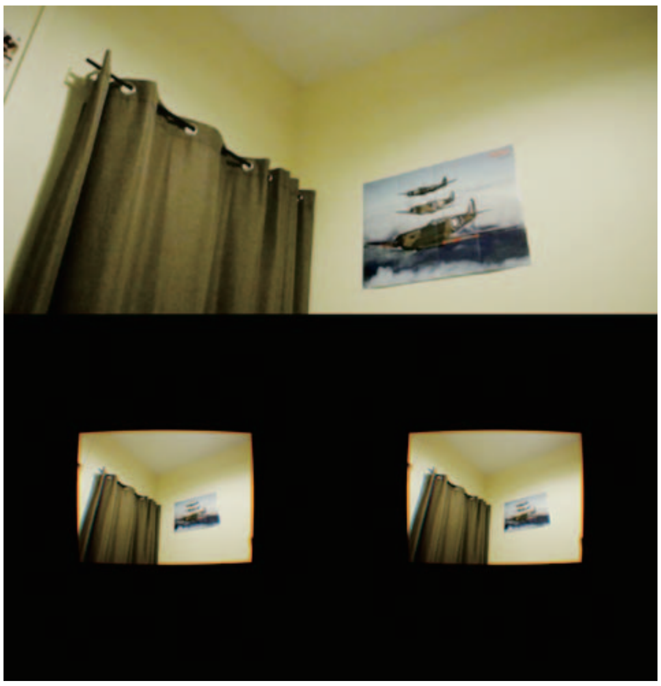


FIGURE 4. View before and after adjustment for VR

By changing the distance to the screen, the image from the webcam filled the users view to a greater extent and gave a realistic image of the world, as shown in Fig. 5.

3.3 Combination for AR

Now that both of the constituent parts were working correctly they needed to be combined together. To do this the second file used for the webcam pass-through continued to be used and edited to also include the LEAP Motion hand from the first file. Some of the set-up process from earlier was repeated by enabling the plug-in in the new file and copying over the previously made game mode to the second file, setting it as the default game mode in this file. This did not however have the same effect as in the previous file. The player spawn in the second file was not a default player spawn location; the player was instead attached to a view port so as the player was not spawned as a default pawn the hands do not appear. In this case, the hands had to be added into the environment manually for them to appear.

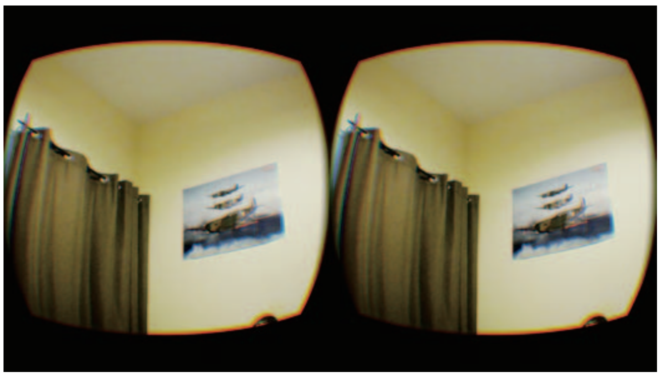


FIGURE 5. Webcam view in the VR headset post screen position adjustment.

To do this, a lesser LEAP hands component was added in front of the camera as shown in Fig. 6. This added the hands in the correct positions in front of the camera. Unlike in the previous instance when the hands were displayed, these hands were created as an item in the level and are persistently there, whereas the previous hands were created by the game mode when the player was spawned at the set player spawn.

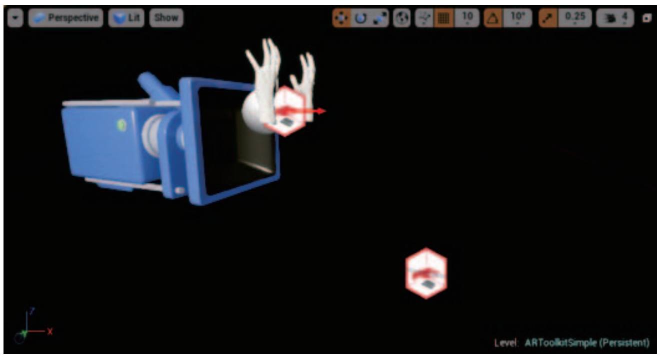


FIGURE 6. A pair of virtual hands placed in the correct position in front of the camera.

Now when the programme was run, the user was able to see their hands inside the headset (Fig. 7) with the full motion that was possible in the level with the LEAP hands on their own, however this was now with a background of the webcam feed creating the required AR.

One final change needed to be made before being able to move onto the software for gesture detection. The software used in Unreal automatically assumed that the LEAP Motion Controller is attached to the front of the headset when testing the programme. However, for the hand tracking to work effectively the LEAP Motion Controller had to be positioned on the desk in front of the user, so the users head movements would not affect the effective palm position compared to the controller and make the robot move unintentionally. To prevent this, the code needed to default to the table top position in all circumstances, and to force it to the do this some code needed to be removed. The 'set static offset' function was edited to remove the transformations and options for the head mounted position, and the handle HMDMode change function was made blank. This meant the code entered the function and then immediately exited. With these two functions edited the code defaulted to the table mounted position regardless of the fact that a head mounted display (HMD) was being used.

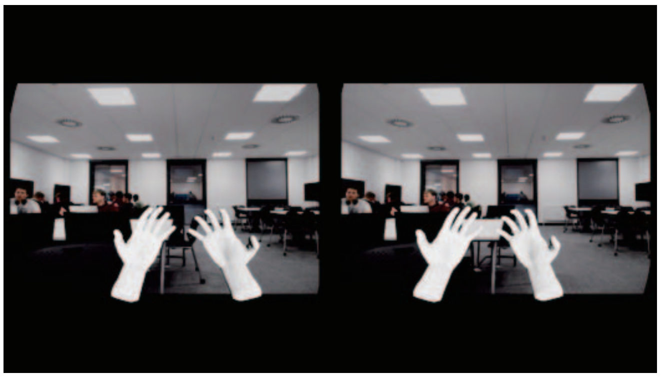


FIGURE 7. The Combination of LEAP Hands and the surrounding environment are now displayed in the headset.

3.4 Hand Position and Gesture Detection

It was decided not to continue using Unreal Engine to track finger positions and gestures. Using direct interaction to the Application Programming Interface (API) through coding meant that the gestures and positions could be made more specific than the built-in gesture blocks in Unreal. It may have been possible to achieve the same goals in Unreal but it would have been much more complicated to use the Blueprint programming method, than to code a new piece of software using a normal language. The code to interact with Baxter would not have been possible in Unreal, so using a separate code to achieve the gestures also removed the problem of having to pass data back and forth between two programmes.

Java was chosen as the language for this second set of code as it is a high-level language making it easier to work with, and it is also a commonly used language. To set up the new coding environment, the Leap Motion Example code, provided by the LEAP Motion SDK, in Java was used as a starting point. This example code handled connecting and disconnecting from the

LEAP Motion, as well as receiving events from the controller. It also gave some example code to show how to detect different hand variables which was a good starting point.

The portion of the code that was edited was the SampleListener class as this was where the data from the Leap Motion Controller is retrieved and could be used. The first section of code set-up variables that would be used later. The most important of these variables were the palm positions for both the left and right hand for the current detected frame, and the frame before that.

If the hand was gripping then the calculations were done on this hand, the first calculation was to find the change in coordinates between frames. The change value found was however in millimetres as this was the unit that the LEAP Motion uses for its values. These values needed to be converted to metres for the change in position to scale correctly when the values are sent to Baxter.

The axis used by the LEAP Motion are orientated differently to the axis used by Baxter. This means that some of the values in the vector had to be moved around so that the direction of the user's hand movement was translated correctly to Baxter's movements. The combination of all these smaller calculations into one large calculation, to take the initial vector to a format usable by Baxter is shown in (1).

Although the vector had been rearranged, the values stored in it were still only changed in position between frames. For the values to be useful they needed to be converted to a co-ordinate that Baxter could move to. To do this the value RCurrentPos was updated. This value was initialised to be $[0.6, 0.2, 0.2]$ which was the default starting position for Baxter's hands. From this initialisation, this variable was continually adjusted with the change in palm position being added to it. This changed the current position vector that Baxter would match.

$$[X_B, Y_B, Z_B] = \frac{[X_{CF}, Y_{CF}, Z_{CF}] - [X_{PF}, Y_{PF}, Z_{PF}]}{1000} + [X_{CP}, Y_{CP}, Z_{CP}] \quad (1)$$

where X_B, Y_B, Z_B defined vector co-ordinate values that were sent to Baxter. X_{CF}, Y_{CF}, Z_{CF} defined the current frame vector co-ordinates. X_{PF}, Y_{PF}, Z_{PF} defined the position vectors from the previous frame. X_{CP}, Y_{CP}, Z_{CP} vector values defined the vector values of Baxter's current arm position.

The three co-ordinates in the $[X_B, Y_B, Z_B]$ vector in (1) were then assigned to their own variables, so they can each be encoded before being sent to Baxter via User Datagram Protocol (UDP).

UDP was achieved using the built in DataPacket features of Java to send the packets. The encoded data of hand trajectories which stored in the buffer variable is then sent to the given port (6101) on the IP address entered. Baxter kinematic controller will then decoded the positional data and handled the movement of Baxter's arms.

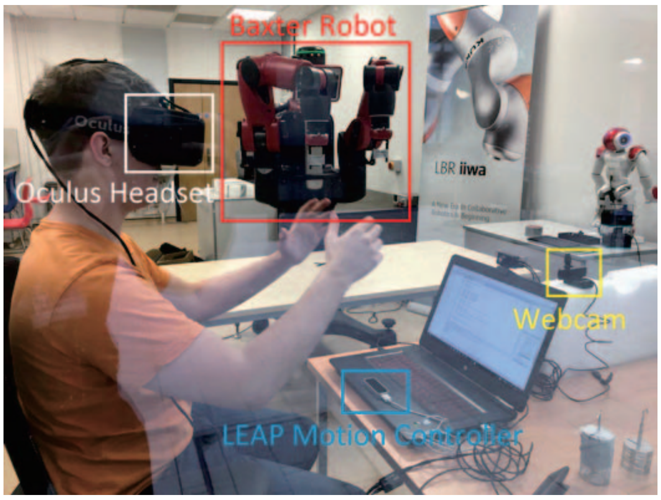


FIGURE 8. Experimental setup.

4 Experiment Result

4.1 Tracking performance

Once all of these steps had been completed the hardware was set up as shown in Fig. 8. When the two programmes were then run, the user sees a view of Baxter and when their hands were raised in front of them saw a virtual model of their hands as shown in Fig 7. When the user clenched their fists, and moved their hands, Baxter's arms changed positions mirroring the user's hands.

The positions of both Baxter's arms and the users hand were tracked and stored, the results of which are shown in Fig. 9.

The LEAP Motion Controller had its limitations. The controller used camera based image recognition tracking, this meant that any hand movement or gesture that obscured part of the hands or fingers could cause errors as the LEAP Motion began to struggle to accurately track parts of the hands. This may have caused gestures that are out of sight of the controller's camera to be picked up incorrectly. It was also possible for the controller to incorrectly position parts of the hand, which caused gestures to be triggered at incorrect times. This was less of a problem with the controller attached to the head mounted display, however that position is only suitable for certain applications and was not able to be used in this case due to

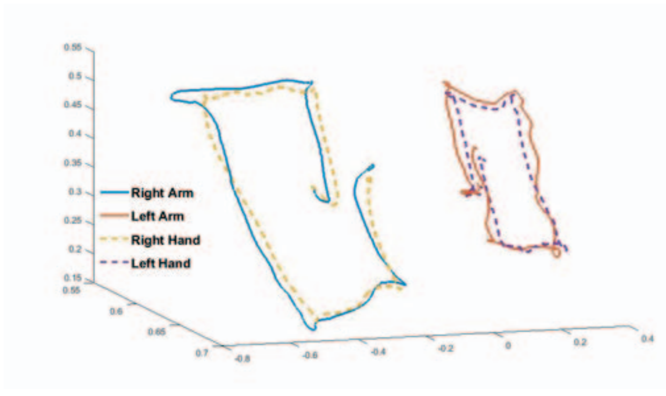


FIGURE 9. Tracking trajectory. Solid lines: actual position trajectories of robot manipulator. Dash lines: commanded position trajectories set by LEAP motion.

the problems with head movement being detected as a change in hand position, as previously discussed.

The surrounding environment also affected the tracking of the Leap Motion as the device uses infrared light for tracking. This meant that infrared light from external sources such as the sun and incandescent bulbs reduced tracking accuracy. The Leap Motion software did detect this and attempted to adjust for it but it is not perfect and tracking accuracy was reduced in certain scenarios.

The LEAP Controllers tracking accuracy was insufficient because it detected small movements but there was a consistent error of +/- 1-3mm. This error can be seen in the results shown in Fig. 9. Baxter's arms did follow the movement of the user's hands and the trajectories are consistent. Baxter's trajectory does vary slightly from that of the original LEAP trajectory. This could be caused by the small tracking error in the LEAP Controller, but the main sources of error are internal uncertainties within the Baxter controller causing slight variations. Differences could also be introduced due to the data rate being sent to Baxter influencing Baxter's reproduction to the co-ordinates.

5 Conclusion

In this paper, a LEAP Motion Controller was used in conjunction with an Oculus Rift VR headset to control the Baxter robot remotely using MR successfully. A user could interact with Baxter by putting on the headset to see a view of the robot from their location, then putting their hands up in front of them. The LEAP Motion Controller then detected these hands and supplied key data about them to a piece of Java software and to Unreal Engine. Unreal Engine rendered a virtual representation of these hands, while the Java software detected if the they were grabbing. If they were, the change in palm position between frames was sent to Baxter's hands via encoded UDP. Using this method, it was possible to replicate the movement of the user's hands with Baxter's hands allowing him to be adjusted remotely.

The results from this paper are positive and show it is possible to use this method to control the Baxter robot. The software could be improved to take advantage of the more advanced gestures that the Leap Motion allows. This would allow more aspects of Baxter to be controlled such as joint rotation and grip. There could also be more features added such as a fine adjustment mode. These adjustments would make this software much more suited for industrial use.

Finally, future work can be done by implementing a stereo camera instead of the 2D camera. While using the 2D camera accurate adjustment was possible, it was much more difficult in some scenarios due to the low resolution 2D image making it hard for the user to get a sense of depth. A more advanced stereo camera would allow the view the user has through the headset to be true 3D, meaning they would have more accurate depth perception. Having depth perception of both their hands and the robot and its background would allow easier and more accurate adjustment by the user, with fewer errors.

References

[1] A. Billard and R. Siegwart. Robot learning from demonstration, 2004.

[2] J. Chen, M. Glover, C. Li, and C. Yang. Development of a user experience enhanced teleoperation approach. In *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 171–177, Aug 2016.

[3] G. Du, P. Zhang, and X. Liu. Markerless human-manipulator interface using leap motion with interval kalman filter and improved particle filter. *IEEE Transactions on Industrial Informatics*, 12(2):694–704, April 2016.

[4] J. C. Garca, B. Patro, L. Almeida, J. Prez, P. Menezes, J. Dias, and P. J. Sanz. A natural interface for remote operation of underwater robots. *IEEE Computer Graphics and Applications*, 37(1):34–43, Jan 2017.

[5] A. Gaschler, M. Springer, M. Rickert, and A. Knoll. Intuitive robot tasks with augmented reality and virtual obstacles. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6026–6031, May 2014.

[6] L. Hamon. Virtual reality and programming by demonstration: Teaching a robot to grasp a dynamic object by the generalization of human demonstrations. *Presence*, 20(3):241–253, June 2011.

[7] S. Livatino, F. Banno, and G. Muscato. 3-d integration of robot vision and laser data with semiautomatic calibration in augmented reality stereoscopic visual interface. *IEEE Transactions on Industrial Informatics*, 8(1):69–77, Feb 2012.

[8] Y. Pititeeraphab, P. Choitkunnan, N. Thongpance, K. Kullathum, and C. Pintavirooj. Robot-arm control system using leap motion controller. In *2016 International Conference on Biomedical Engineering (BME-HUST)*, pages 109–112, Oct 2016.

[9] L. Pries. New production systems and workers' participation–a contradiction? some lessons from german automobile companies. *Charron, E. & Stewart, P.(Hg.), Work and Employment Relations in the Automobile Industry, New York*, 2004.

[10] Y. Sung and K. Cho. Collaborative programming by demonstration in a virtual environment. *IEEE Intelligent Systems*, 27(2):14–17, March 2012.

[11] J. Wang, Y. Li, and X. Zhao. Inverse kinematics and control of a 7-dof redundant manipulator based on the closed-loop algorithm. *International Journal of Advanced Robotic Systems*, 7(4):1–9, 2010.

[12] C. Yang, P. Liang, Z. Li, A. Ajoudani, C. Y. Su, and A. Bicchi. Teaching by demonstration on dual-arm robot using variable stiffness transferring. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1202–1208, Dec 2015.

[13] C. Zieliński. *Robot programming methods*. Oficyna Wydawnicza Politechniki Warszawskiej, 1995.