Tele-operating Spot's Robotic Arm Using Mixed Reality

William Talbot, Alexandru Ţop, Cailin Ringstrom, Emilia Szymanska
Eidgenössische Technische Hochschule Zürich
Robotics, Systems and Control MSc Program

Supervisors: Boyang Sun, Marcel Geppert
Eidgenössische Technische Hochschule Zürich
Computer Vision and Geometry Lab

Abstract

As mixed reality technology has matured in quality and popularity over the last decade, a demand for sophisticated and immersive applications has risen. Similarly intelligent robots with advanced autonomous capabilities are increasingly in demand to solve complex, dangerous or onerous tasks, and are beginning to become part of our everyday lives. Given this context, the project described in this report presents a system that bridges the gap between mixed reality and advanced mobile robotics. Using a Microsoft Hololens 2 or Android device, a human user can intuitively control the robotic arm attached to a Boston Dynamics Spot robot, in order to facilitate more sophisticated interaction with the environment or enhanced perception applications. Quantitative tests and user experience surveys complement the system design presented in this report, to evaluate the system's intuitiveness and immersion, and drive the discussion of possible development directions.

1. Introduction

Mixed Reality (MR), which seeks to seamlessly blend the digital and the real-life world, is an increasingly popular option for complex tasks requiring high user interaction [5, 24]. Alongside virtual and augmented reality (VR and AR) [6], it offers opportunities for countless applications in fields such as medicine [14], education [11, 28], design [15], infrastructure and many others [2, 27]. The integration of mixed reality solutions with robotic systems is a highly investigated topic [4, 9, 13]. For robot teleoperation, many remote controllers are not intuitive, have a steep learning curve, or require users to control multiple joints simultaneously. On the other hand, a mixed reality device – which offers a customizable, multimedia user interface in addition to head, eyes and hands tracking – gives a wide range of opportunities for remote control implementations. Therefore, the project presented in this report takes advantage of the features available in a MR headset to explore the teleoperation of a robotic arm attached to a legged robot. The device chosen for this purpose was Microsoft's Hololens 2 [22], which is supported by the Mixed Reality Toolkit (MRTK) to facilitate the development of MR applications in Unity. The robot used in this study was Boston Dynamic's quadruped Spot, extended with a 6 degree-of-freedom robotic arm. Starting with simple head-motion tracking and end effector motion, the objective of the project was to implement a more advanced MR-based operational pipeline enabling the user to control Spot's arm and body position in accordance with the head movement and voice commands. In addition to this, images from the arm's high resolution camera are streamed to the Hololens display so that the user shares the perspective of the end effector. As in similar applications [29], an intuitive interface design is essential for a positive user experience. Thus the system was evaluated by collecting qualitative feedback after demonstrations of the system to new users. This was in addition to several quantitative studies examining the accuracy and latency performance of the system, detailed in Section 4.

2. Related work

There exist various approaches to robotic arm teleoperation to address a broad spectrum of diverse problems. One solution to the teleoperation problem involves decoding grasp gestures from electromyographic signals [3], which then can be mapped to robotic arm movements [10]. The EMG-based control can be further improved by integrating acceleration measurements [19]. Another approach to robot arm remote control focuses on readings from sensory gloves [8, 12]. However, more and more systems rely on image streams from either depth or conventional color cameras [1, 20]. This trend offers plenty of opportunities for integrating mixed reality device with robotic systems for teleoperation and collaboration. Some research has al-

ready been conducted in this field, mostly with the Oculus Rift [16] virtual reality device and with Microsoft Hololens devices [17, 18, 21].

Many applications have already been made for the Microsoft HoloLens (1) and Hololens 2 [23] using the inbuilt "Research Mode", enabling access for computer vision capabilities [26]. However, an application for remotely operating a robotic arm mounted on a quadruped robot was not found. The the most similar project relating to Spot and Hololens integration found was a proposal to combine spatial computing and egocentric sensing on mixed reality devices to improve collaboration between humans and robots [7]. One of the use cases presented in this work was to plan a mission for Spot by placing holographic markers in the augmented reality environment. The authors also implemented teleoperation of a UR5 robotic arm with hand-like end effector. However, the combination of these two use cases was not explored, and no research examined the concept of using head tracking for control of robot movement, only hand tracking for end effector manipulation. Therefore, the work presented in this report hopes to provide a modular foundation for whole robot teleoperation which is lacking in existing research, and stimulate further investigation into the integration of mixed reality with arm-equipped mobile robots.

3. System Overview

This project sought to build a general, robust and modular foundation for future application-oriented robot teleoperation activities. It's contributions are therefore:

1. Spot interface which reinterprets incoming poses and sends both arm and body motion commands to Spot such that the arm end effector realizes the requested pose.

2. Hololens 2 application which performs head tracking, and provides an immersive audio-visual interface for the teleoperation of a robot.

3. Unity-to-ROS pose conversion module which converts from the left-handed y-up coordinate system of native Unity applications to the right-handed z-up convention of the ROS standard [25].

4. Improvements to Spot ROS wrapper, spot_ros, including bug fixes and adding a dynamic arm movement planning time option.

5. A demonstration and quantitative evaluation of a complete system that integrates the above modules.

To achieve the integration of two independent robotic systems, the Robotic Operating System (ROS) middleware was used as a communications layer. ROS is an open-source suite of libraries and tools that facilitates inter-robot communications through three separate mechanisms, all of which are used in this project. The first is a publish-subscribe pattern whereby processes called *nodes* can either send (publish) or listen to (subscribe) well-defined message packets on *topics*. For example this is used in this project for the sending of head-track and reset pose messages, and the reception of the image stream from the Spot arm end effector. The second is a server-client service protocol which allow for a blocking call of a function in another process, again through well-defined ROS message packets. Many of the commands in the spot_ros driver are implemented as services, such as opening or closing the gripper, sitting or standing, powering on or off, and using the arm inverse kinematic controller. Services are particularly useful for sending commands as they return response information such as indicating the success or validity of the command. This project makes extensive use of these services. The final mechanism available in ROS is actions, which like services allow for commands and provide feedback, but are non-blocking so are designed for operations which take a substantial amount of time to complete. Since the motion of the Spot body is not instantaneous, this project's spot interface makes use a ROS action for tracking the completion of spot body trajectory commands. Since ROS is primarily designed for `C++` and `python` applications, this project makes use of two tools provided by Unity, a ROS `C#` API Unity-Robotics-Hub, and ROS-TCP-Endpoint which is required for both ROS message transmission and reception from the Hololens 2 device.

A comprehensive overview of the modular system is illustrated in Figure 1, with the notable omission of the ROS-TCP-Endpoint node for clarity, required for inter-device network communication. In addition to the aforementioned modules is an image compression node from `image_transport`, a widely used ROS package, discussed in Sections 3.2 and 4.

3.1. Spot robot

The Spot interface is responsible for issuing commands to Spot so that the arm end effector motion mirrors that of an incoming pose stream, which in this project originates from a Hololens 2 or Android mobile device. To achieve this, the interface needs to decompose these desired poses, in the form of ROS `geometry_msgs/Pose` messages, into two. One of these poses is issued to the arm inverse kinematic controller, and the other commands the Spot body itself. This process is complicated by three additional behaviors which improve the user experience and fluidity of the motion. These are:

1. User-configurable arm initial position and orientation.

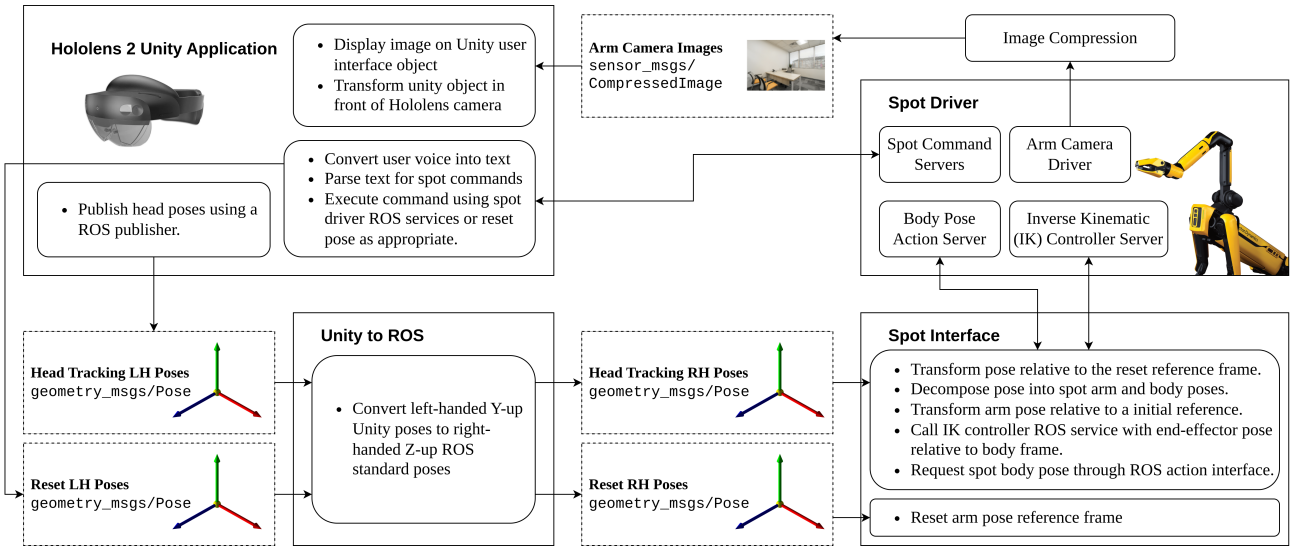

Figure 1. System Overview: This project integrates three new modules as well as publicly available ROS and Unity APIs to achieve immersive Spot teleoperation. These are the Hololens 2 Unity application, a mixed reality audio-visual interface, the Spot interface, and a Unity-to-ROS conversion node.

2. Restriction of end effector position within a bounding box region around the initial position, where the behavior of the inverse kinematic controller is well-defined.

3. On-demand reset feature, whereby the user can reset the arm pose so that future motion is relative to the current pose of the Hololens 2 or mobile device.

Furthermore, two spot motion modes were implemented. The first is a static mode, where Spot itself remains motionless and only the arm is controlled, constrained within the dimensions of the bounding box. The second extends on this, by incorporating translation of the Spot base so that x and y dimensions of the planning space are fully realizable. The complexity of this system requires an understanding of the relevant coordinate frames of the robot, shown in Figure 2. The notation T_A^B is used to denote the frame $\{B\}$ in the reference frame of $\{A\}$, or equivalently the $SE(3)$ transform from $\{A\}$ to $\{B\}$. The direction of transforms are reversed by taking the inverse, so $T_B^A = \left(T_A^B\right)^{-1}$. Using Figure 2 as a reference, the origin of the Hololens application is defined as $\{O_A\}$, which may differ from the internal origin tracked by the Hololens device when it is powered on, denoted as $\{O_H\}$. When the application is started, this frame is fixed to the pose of the Hololens at that time, $T_{O_H}^{O_A} = T_{O_H}^H$ with $T_{O_A}^H = I$. When the user sends a reset command (through voice commands, see Section 3.2), the application origin is updated to the current pose, $T_{O_H}^{O_A} = T_{O_H}^H$, just as at start-up.

Similarly to the Hololens 2, Spot also has two origin reference frames. Internally, Spot maintains an odometry frame, labeled $\{O_S\}$ for Spot origin in this project, and measures the the transform $T_{O_S}^B$ of the body frame $\{B\}$ in this frame through visual and kinematic odometry algorithms. Because the mirrored motion must be relative to the starting pose of Spot, this project defines the body origin frame $\{O_B\}$. The initial end effector frame $\{I\}$ is defined with respect to the body frame as T_B^I, and the transform from $\{I\}$ to the end effector frame $\{E\}$ is T_I^E

The key idea for mirroring the user's motion then is to map the transform between the application origin and Hololens 2, to the transform of the Spot body from its origin and the end effector from its initial pose. This can be expressed as Equation 1.

$$T_{O_A}^H = T_{O_B}^B T_I^E \tag{1}$$

The method to achieve this equality is a process of several steps:

1. When the Spot interface ROS node is launched, look up the transform $T_{O_S}^B$ of the Spot body from its odometry frame, and set $T_{O_S}^{O_B}$ to it so that $T_{O_B}^B = I$.

2. As discussed previously, when the Hololens application is launched, set $T_{O_H}^{O_A} = T_{O_H}^H$. This transform is adjusted whenever the user sends a reset command.

3. When a new head track pose $T_{O_H}^H$ is received, the pose relative to the user application origin is computed as $T_{O_A}^H = T_{O_A}^{O_H} T_{O_H}^H = \left(T_{O_H}^{O_A}\right)^{-1} T_{O_H}^H$.

4. The new end effector transform is computed as $T_I^E = \left(T_{B_O}^B\right)^{-1} T_{O_A}^H$ (from Equation 1). This is an optimistic

transform, which may not lie within the bounding box region.

5. Set the translation component of T_I^E to the closest point within the bounding box.

6. If in translation mode, compute the new Spot body position as $T_{O_B}^B = T_{O_A}^H \left(T_I^E\right)^{-1}$ (from Equation 1), which will only change if T_I^E lay outside the bounding box in the previous step. Send $T_{O_B}^B$ to the Spot driver as a command for the Spot body motion. If in static mode, skip this step.

7. Compute the end effector pose with respect to the body frame as $T_B^E = T_B^I T_I^E$, and send it as a command to the inverse kinematic arm controller.

The Spot driver expects all poses to use a right-handed z-up reference frame convention as described by the ROS standard [25], however Unity uses a left-handed y-up convention. Therefore a Unity-to-ROS pose conversion node was implemented to convert between the incompatible formats. This is shown in Figure 1.

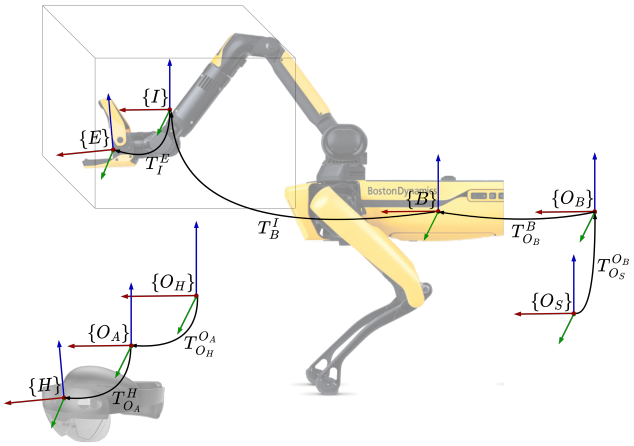

Figure 2. Visualization of the coordinate frames of the Spot robot and Hololens 2 device. The bounding box region for allowed end effector positions is also shown.

3.2. Hololens 2 Device

Unity was used in order to develop an application for the Hololens for the user to interact with Spot. In order to communicate with the rest of the system, the Unity-Robotics-Hub API was used. The three main functionalities of the Hololens application are:

1. Tracking and streaming the head position of the Hololens user.

2. Receiving images from the Spot end effector camera and displaying them in real time in a User Interface (UI), as presented in Fig. 3.

3. Interacting with the Spot robot with voice commands.

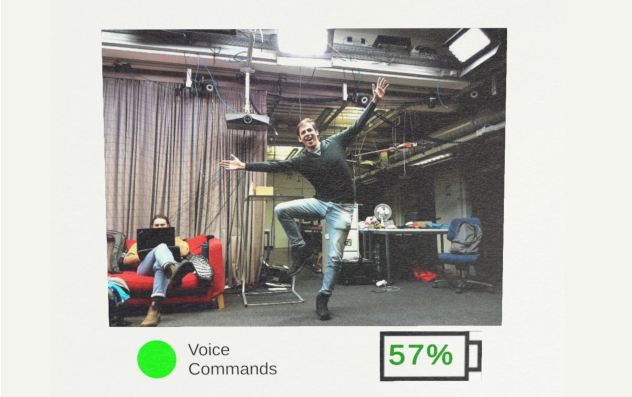

Figure 3. Hololens User Interface with hidden command list.

To achieve head tracking, the current position of the Hololens is read and published as a ROS `geometry_msgs/Pose` at the frame rate of the Hololens. All computations to map this to a ROS coordinate frame convention are left to the Unity-to-ROS converter as described in Section 3.

Images from the robot arm end effector are published as `sensor_msgs/Image` messages by the Spot ROS driver. An option was also added to receive compressed images as `sensor_msgs/CompressedImage` messages in effort to reduce network latency. This requires using an additional ROS plugin called image_transport which converts from the `sensor_msgs/Image` message type to the `sensor_msgs/CompressedImage` message type. Based on user feedback, a white background was added to help separate the images from the background, to improve the visibility of the image stream and immersion of the experience.

In addition to the core motion control functionality, voice commands to control Spot were added (Fig. 4). The text-to-speech functionality of the Hololens was also used in order to give audio feedback to the user, informing them on the success of their commands. To remind the user of the available options, a screen is shown on startup which lists all available commands and their function. This screen can also be hidden and shown using voice commands. To help prevent unintentional commands from being sent to the robot, speech commands are initially disabled, and can be enabled and disabled by the user. All requests are sent to Spot via ROS services provided by the Spot ROS wrapper, as detailed in Section 3.

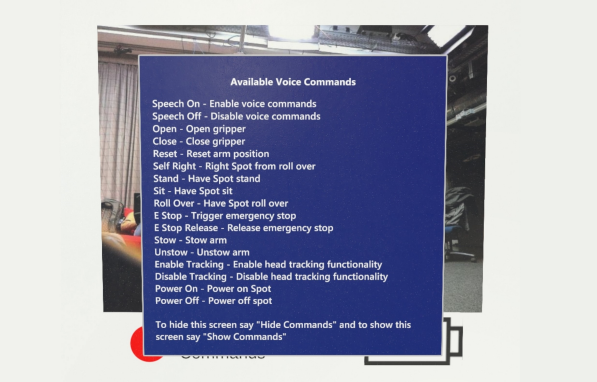

Figure 4. Hololens User Interface with the command list.

Finally, a automatically updating battery icon and an voice command indicator light, showing if speech is enabled or disabled, were also added to the UI for an improved user experience.

3.3. Android device

The MRTK Unity application, originally built for Hololens 2, was also deployed on Android with the help of Google ARCore library. The position of the objects was adjusted to make the interface easier to view on an Android phone. Deployment to the android device was highly advantageous during development. The variety of devices allowed for easier and faster testing of our application, facilitating the simultaneous development of modules in our system. Furthermore the deployment process for Android devices is much faster than that for the Hololens 2, allowing us to more rapidly test new features and resolve issues. A screenshot of the application can be seen in Figure 5.

Figure 5. Spot controlled with Android app.

The Android application deployment required the modification of the shader used for rendering the application as the latest version of MRTK is not fully compatible with the Google ARCore. The issue was identified thanks to the build errors given by Unity and was fixed using a workaround provided in a Github issue. These issues should dissipate as the compatibility between the libraries improves in further version releases.

Datatype	Bandwidth (KB/image)
Uncompressed	920
Compressed	82

Table 1. Network bandwidth measured when streaming compressed and uncompressed images from Spot to the Hololens 2.

A number limitations were identified for the Android application:

- **Voice Recognition** – MRTK Voice Recognition Input is not currently compatible with Android devices and Google ARCore does not provide voice capture functionality. A custom module would need to be implemented in order to support voice recognition input on Android devices.
- **MRTK Buttons** – MRTK Buttons have issues when used on Android deployments. While both the animation and sound of the buttons triggers on press, unfortunately the attached scripts do not execute as they do when the app is deployed to Hololens 2 or when the application was run with Unity. This issue was not further investigated and the feature was discarded.

Due to the presented limitations, the Android application was only used in part of the testing of the spot's gripper and body movement. Without Voice Recognition or Buttons, the app was not appropriate for the complete testing of the Spot control interface, such as the reset-on-demand functionality.

4. System Evaluation and Experiments

4.1. Image Streaming Evaluation

Bandwidth and latency were measured for streaming images from Spot to the Hololens as both compressed and uncompressed images, with the results for both cases shown in Table 1. It was observed that using compressed images resulted in an 11.25-fold network bandwidth decrease, which is a substantial improvement and may be highly advantageous if this system were deployed within a bandwidth-limited environment.

To measure the latency performance and assess the benefits of image compression in this system, we measured how delayed image messages were at different stages of the pipeline. For each message type, two measurements were made: the difference between the image timestamp and the Spot system time at arrival time and the difference between the image timestamp and remote device (Hololens

	Uncompressed	Compressed
Hololens Latency (s)	0.717 ± 0.153	0.653 ± 0.160
Onboard Latency (s)	0.586 ± 0.153	0.590 ± 0.152
Network Latency (s)	0.127 ± 0.032	0.067 ± 0.032

Table 2. Comparison of mean latency of arm image stream for compressed and uncompressed messages over a 60 s period. The image delay is measured on the Hololens device and onboard Spot itself, and the network latency is the difference between them.

2, Android device, ground station) system time. These measurements are summarized in Table 2. It was observed that compression decreased the overall delay by 0.06 s, which can be attributed to a reduction network latency due to the reduced message size. However, the baseline latency of image acquisition on the Spot system itself at almost 0.6 s significantly outweighs the network latency, and thus comparatively image compression does not have a large effect on the user experience.

4.2. Spot Movement Evaluation

The Spot driver publishes the relative transform between the arm links using the joint states at a high frequency. From this the pose of the end effector relative to the base link is estimated, and it is thus possible to measure the error between the requested and true pose. Figure 7 shows an example of the position and angular error experienced during a live demonstration of static Spot teleoperation with the Hololens 2 (movements presented in Fig. 6). Through an analysis of these results with a video of the demonstration, several clear sources of error could be identified. The key source of the persistent error across the whole recording is latency in the response, caused primarily by the planning and execution time of the arm inverse kinematic controller. Large movements at higher velocities accentuate this, such as between 60 and 63 s and between 68 and 70 s. A second source of error can be seen in the 71 to 73 s period, where the translation error increases significantly. This occurs because the user's head moves far enough from their origin that the end effector reaches the edge of the bounding box operating region. This error is therefore expected as part of the interface design. Finally the large angular errors in the 52 to 54 s and 80 to 85 s periods is a consequence of the significant limitation of the inverse kinematic controller to yaw (pure rotation about z axis) when the user rotates their head from side to side. The joint configuration of the arm caused the simple arm controller to fail for this movement. Since this is a very natural movement for a new user and the large angular error results in a camera stream that does not reflect their head motion, this issue posed a serious risk of breaking the immersion of the teleoperation experience. While this could be mitigated by teaching new users to look down or up while turning their head, a movement pattern that the arm controller could smoothly achieve, ultimately the problem could only be solved by using a different robot arm or more sophisticated planner.

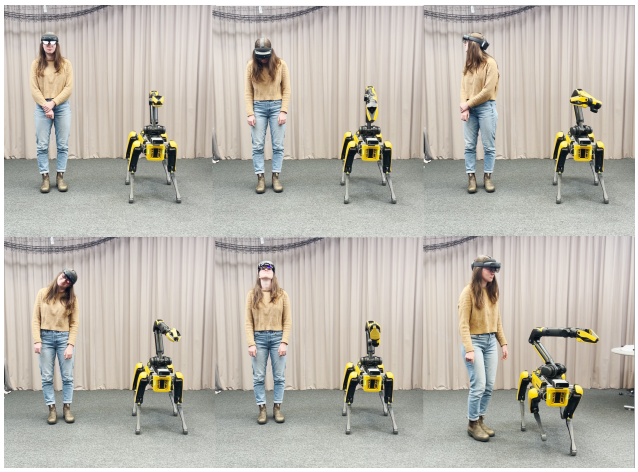

Figure 6. Examples of robot control.

4.3. User Experience Testing

Four participants were recruited for user experience tests, three of whom had not previously used a Hololens device. We had all participants perform the initialization procedure of Spot using voice commands and control Spot's movement by moving around using the Hololens. Following the test, all completed a survey with giving both numerical ratings and open-ended feedback on different components of the system. Figure 9 contains charts showing the ratings users gave for the image streaming, voice commands, and Spot movement. Figure 8 shows a pie chart for the number of unintended events they experienced while using the application. Unintended events include voice recognition errors such as not detecting the spoken command or calling a different command than the one intended by the user. Wrong behavior of the robot while following the user is also considered unintended event.

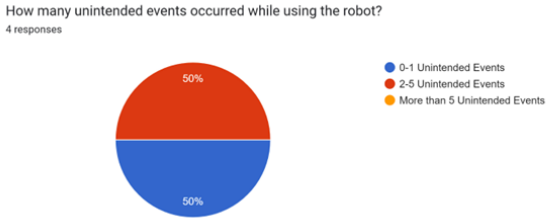

Figure 8. Chart showing the number of unintended events experienced by each user.

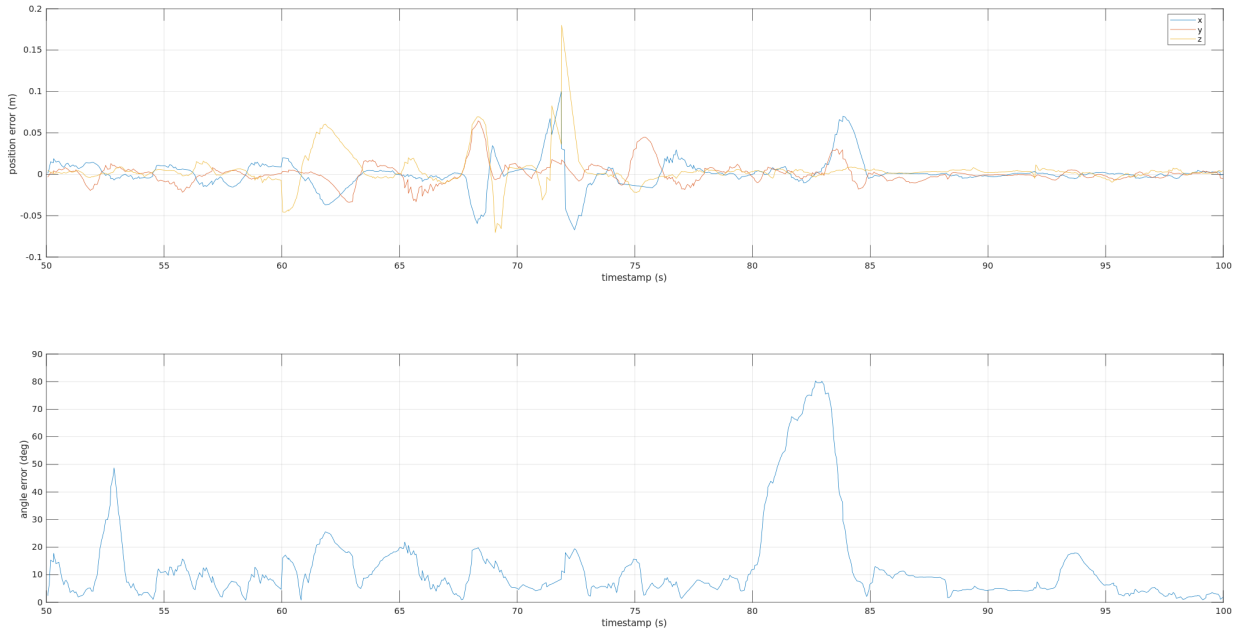

Figure 7. End effector translational and angular error during a static live demonstration.

Open-ended questions asked users what they liked, did not like, and what they would like to see added. Common positive feedback included responsiveness and smoothness of the system. Negative feedback primarily focused on the image stream with complaints about blurriness, lack of visibility against the background, and video delay. The most common suggestions for future work were adding body rotation and improving the yaw rotation of the end effector.

5. Discussion

Throughout the development of the project and through user experience testing, many challenges were encountered and avenues for improvement were identified. The most common improvements and additions suggested from the user feedback were improved yaw motion of the arm, improved image stream quality, improved responsiveness, and the addition of body rotation. Prior to user experience testing, we have also identified these issues and identified potential ways to improve them.

For the improvement of arm yaw movement arm and the addition of body rotation, additions would need to be made to the controller. Currently, the built-in inverse kinematic controller is significantly limited in yaw rotation. As stated earlier in the report, in order to improve this, a more sophisticated controller would need to be implemented, or arm with a different joint configuration be used.

Furthermore, the Spot arm planner currently plans every movement with a constant time for pose completion. Both faster and slower user movement would be planned with the same execution time. A short time would not be viable for faster user movements as they could be time limited and fail while a short time would not be viable for slower user movements as they would have a high latency and making the arm lag heavily behind the user. An improvement for addressing the issue would be setting a determined planner time which would be dependent on the speed of the user movement.

It would also enhance the functionality of the system if the Spot body rotation was implemented. As for now, Spot only translates its body to compensate for the user's out-of-bound movements. That limits the applications of the system as it is almost impossible to see what happens behind the robot – the arm limits do not allow for 360^o rotation around the Z axis of the first joint, so to look behind, a user would need to make the robot translate to the side and then try to see what was behind the robot. This is an inconvenient workaround, therefore a better solution including Spot body rotation would need to be implemented.

The Spot ROS wrapper also proved to be somewhat problematic. The arm control service always failed, and although the requested movement was performed, it is believed that this failure may have been causing the arm to vibrate under certain joint configurations. This was solved by identifying and correcting the error in the source code. Furthermore, the maximum linear and angular velocities of Spot's movement were set to 0 by default, so the movement commands were not executed and no useful feedback for debugging this failure were reported at the software level.

It was therefore essential to remember to run the launch file with non-zero velocities parameters. Additionally it was noticed that a patch for Spot body movement had been implemented in the available ROS action interface, but not topic interface, so a transition was made to the action interface.

With regards to image stream quality, improvements in responsiveness and visibility were suggested in the user feedback. As can be seen in Table 2, we were able to improve the latency using compressed images, but the delay between image capture and receiving the ROS message output from the Spot ROS driver far outweighs the latency from transmitting the image over the network. Therefore, the improvements that can be made here are limited, as they probably depend on implementation details in Spot ROS wrapper or Boston Dynamics API. However, one way to improve the visibility and immersivity of the application could be by experimenting with VR applications. The advantage of VR applications is that they offer separation from the real environment which could make the application more immersive than using an AR device in this case. When the Hololens was present in the same environment as Spot, the overlay of the camera stream on the background environment could be distracting or confusing for the user thus breaking the immersion.

Another issue encountered during the development of the application was the limitation of MRTK features in Android applications. As discussed in Section 3.3, the use of buttons was problematic and the voice recognition did not work at all, requiring a custom module to be built, and limiting its usefulness in comparison to the full-functioning Hololens 2 application. This issue could be resolved by using a different tool for programming the app, at the cost of making the parallel development of Hololens and Android app more complex and time consuming.

6. Conclusions

Mixed reality and robotic technologies are becoming increasingly powerful and prevalent, and their integration is an opportunity for intuitive human-robot interaction. The aim of this project was to explore this idea by developing an application enabling a Hololens 2 user to control the pose of a Spot arm end effector with their head motion while viewing the image stream from the gripper camera on their display. In addition to this, further control capabilities were added into the application through voice recognition tools.

Long-term, there are many real life human-in-the-loop applications for which the work of this project could serve as a foundation, such as remote surveillance and inspection. This mixed reality application allows a user to be immersed in and explore a remote environment from any offsite location. The benefits of this are potentially significant, such as reducing risk to human health in potentially dangerous sites, to cost reduction since inspection experts do not need to travel physically to the site of interest.

In additional to those mentioned in Section 5, future steps could including testing and comparing performances on different AR and VR headsets (e.g. Oculus Rift, Magic Leap One or Raptor AR). It would also be interesting to apply and compare the implemented approach to different kinds of robots with robotic arms, such as bipedal robots and investigate how they could be utilized in real-life applications.

This report has demonstrated the potential for MR technology to provide an immersive and intuitive method for controlling the pose of a robot end effector. The proposed application could be used as a foundation for further projects in the field of human-robot interaction, remote surveillance, infrastructure inspection, and many other fields.

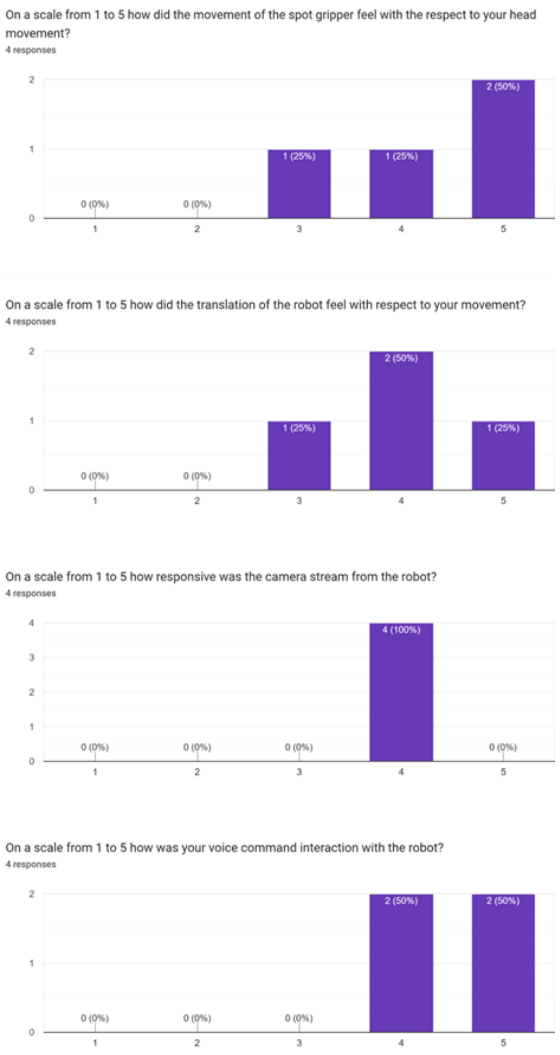

Figure 9. Charts of user experience ratings.

References

[1] Mohamed O. Alamin, EssamEldin M. Khadir, and Sharief F. Babiker. A vision-based teleoperation method for a robotic arm with 4 degrees of freedom. In *2016 Conference of Basic Sciences and Engineering Studies (SGCAC)*, pages 19–23, 2016. 1

[2] Lisa Avila and Mike Bailey. Augment your reality. *IEEE Computer Graphics and Applications*, 36:6–7, 01 2016. 1

[3] I. Batzianoulis, S. El-Khoury, E. Pirondini, M. Coscia, S. Micera, and A. Billard. Emg-based decoding of grasp gestures in reaching-to-grasping motions. *Robotics and Autonomous Systems*, 91:59–70, 2017. 1

[4] Ian Yen-Hung Chen, Bruce MacDonald, and Burkhard Wunsche. Mixed reality simulation for mobile robots. In *2009 IEEE International Conference on Robotics and Automation*, pages 232–237. IEEE, 2009. 1

[5] Steven Chen and Henry Duh. Interface of mixed reality: from the past to the future. *CCF Transactions on Pervasive Computing and Interaction*, 1:1–19, 01 2019. 1

[6] Pietro Cipresso, Irene Alice Chicchi Giglioli, Mariano Alcañiz Raya, and Giuseppe Riva. The past, present, and future of virtual and augmented reality research: A network and cluster analysis of the literature. *Frontiers in Psychology*, 9, 2018. 1

[7] Jeffrey Delmerico, Roi Poranne, Federica Bogo, Helen Oleynikova, Eric Vollenweider, Stelian Coros, Juan Nieto, and Marc Pollefeys. Spatial computing and intuitive interaction: Bringing mixed reality and robotics together. *IEEE Robotics & Automation Magazine*, 29(1):45–57, 2022. 2

[8] Bin Fang, Di Guo, Fuchun Sun, Huaping Liu, and Yupei Wu. A robotic hand-arm teleoperation system using human arm/hand with a novel data glove. pages 2483–2488, 12 2015. 1

[9] Fabrizio Ghiringhelli, Jérôme Guzzi, Gianni A Di Caro, Vincenzo Caglioti, Luca M Gambardella, and Alessandro Giusti. Interactive augmented reality for understanding and analyzing multi-robot systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1195–1201. IEEE, 2014. 1

[10] Hussein F. Hassan, Sadiq J. Abou-Loukh, and Ibraheem Kasim Ibraheem. Teleoperated robotic arm movement using electromyography signal with wearable myo armband. *Journal of King Saud University - Engineering Sciences*, 32(6):378–387, 2020. 1

[11] Jennifer Herron. Augmented reality in medical education and training. *Journal of Electronic Resources in Medical Libraries*, 13:1–5, 05 2016. 1

[12] Haiying Hu, Jiawei Li, Zongwu Xie, Bin Wang, Hong Liu, and G. Hirzinger. A robot arm/hand teleoperation system with telepresence and shared control. volume 2, pages 1312 – 1317, 08 2005. 1

[13] Wolfgang Hönig, Christina Milanes, Lisa Scaria, Thai Phan, Mark Bolas, and Nora Ayanian. Mixed reality for robotics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5382–5387, 2015. 1

[14] Karl-Franz Kaltenborn and O Rienhoff. Virtual reality in medicine. *Methods of information in medicine*, 32:407–17, 03 1994. 1

[15] Lee Kent, Chris Snider, James Gopsill, and Ben Hicks. Mixed reality in design prototyping: A systematic review. *Design Studies*, 77, 09 2021. 1

[16] Maram Khatib, Khaled Al Khudir, and Alessandro De Luca. Human-robot contactless collaboration with mixed reality interface. *Robotics and Computer-Integrated Manufacturing*, 67:102030, 2021. 2

[17] Tomas Kot, Petr Novak, and Ján Babjak. Using hololens to create a virtual operator station for mobile robots. pages 422–427, 05 2018. 2

[18] Ondrej Kyjanek, Bahar Al Bahar, Lauren Vasey, Benedikt Wannemacher, and Achim Menges. Implementation of an augmented reality ar workflow for human robot collaboration in timber prefabrication. 06 2019. 2

[19] Juxi Leitner, M. Luciw, Alexander Förster, and J. Schmidhuber. Teleoperation of a 7 dof humanoid robot arm using human arm accelerations and emg signals. 06 2014. 1

[20] Shuang Li, Jiaxi Jiang, Philipp Ruppel, Hongzhuo Liang, Xiaojian Ma, Norman Hendrich, Fuchun Sun, and Jianwei Zhang. A mobile robot hand-arm teleoperation system by vision and imu, 2020. 1

[21] Congyuan Liang, Chao Liu, Xiaofeng Liu, Long Cheng, and Chenguang Yang. Robot teleoperation system based on mixed reality. In *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 384–389, 2019. 2

[22] Yang Liu, Haiwei Dong, Longyu Zhang, and Abdulmotaleb El Saddik. Technical evaluation of hololens for multimedia: A first look. *IEEE Multimedia*, PP:1–1, 10 2018. 1

[23] Sebeom Park, Shokhrukh Bokijonov, and Yosoon Choi. Review of microsoft hololens applications over the past five years. *Applied Sciences*, 11:7259, 08 2021. 2

[24] Somaiieh Rokhsaritalemi, Abolghasem Sadeghi-Niaraki, and Soo-Mi Choi. A review on mixed reality: Current trends, challenges and prospects. *Applied Sciences*, 10:636, 01 2020. 1

[25] ROS. Standard units of measure and coordinate conventions. Available at https://www.ros.org/reps/rep-0103.html. 2, 4

[26] Dorin Ungureanu, Federica Bogo, Silvano Galliani, Pooja Sama, Xin Duan, Casey Meekhof, Jan Stühmer, Thomas J. Cashman, Bugra Tekin, Johannes L. Schönberger, Pawel Olszta, and Marc Pollefeys. Hololens 2 research mode as a tool for computer vision research, 2020. 2

[27] Rick Van Krevelen and Ronald Poelman. A survey of augmented reality technologies, applications and limitations. *International Journal of Virtual Reality (ISSN 1081-1451)*, 9:1, 06 2010. 1

[28] Shiyao Wang, Michael Parsons, Jordan Stone-McLean, Peter Rogers, Sarah Boyd, Kristopher Hoover, Oscar Meruvia-Pastor, Minglun Gong, and Andrew Smith. Augmented reality as a telemedicine platform for remote procedural training. *Sensors*, 17:2294, 10 2017. 1

[29] Helene Xue, Puneet Sharma, and Fridolin Wild. User satisfaction in augmented reality-based training using microsoft hololens. *Computers*, 8:9, 01 2019. 1