ReVRSR: Remote Virtual Reality for Service Robots

Amel Hassan, Ahmed Ehab Gado, Faizan Muhammad

March 17, 2018

Abstract

This project aims to bring a service robot's perspective to a human user. We believe that developing and using this technology would contribute to a deeper understanding of a robot's state during software and hardware development of these robots as well as be a potential platform for a range of different applications such as telepresence robots, remote robot tours, remote robot debugging and robot monitoring.

For the scope of this project, we will use a TurtleBot2 with an Oculus Rift to provide a stereoscopic view through the TurtleBot2's 3d camera across a wireless network to a human user. We will also provide the user control over the TurtleBot2's movements which will further enhance the VR experience.

1 Introduction

Within the current state of the art in robotics, there is a huge range of functionality that human programmers and users can exploit in a service robot. But we take a different approach and try to answer the question; what if you could experience being the robot itself?

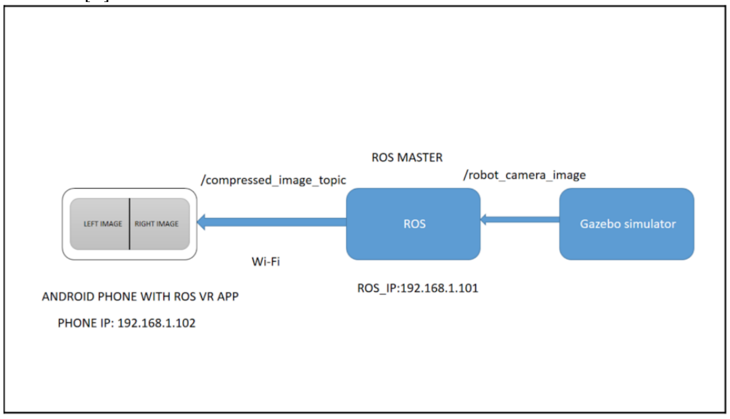
We explore the possibility of augmenting human intelligence with autonomous robots to do what can not currently be done individually by either entity on their own. By giving a human user a virtual perspective of the robot along with dynamic control of it, we believe we can prototype an exciting new technology that can be used in a variety of unique fields, from education to industrial work and from surgery to leisure.

2 Related Works

In his book ROS Robotics Projects, Lentin shares a variety of robotics projects that use ROS to see, sense, move, and perform other various functions [2]. In particular, he has written about a ROS project that teleoperates a robot using a VR headset and leap motion. Similar to our proposed project, he used a VR headset system to experience the environment of the TurtleBot. Although he mentions that he used a "cheap VR headset", in particular Google Cardboard, we hope to use Oculus, a more advanced system, which will allow for a smoother transmission of data to a from ROS. In addition, while controlling the robot using hand gestures is not one of our initial goals, Lentin provides us with an idea of how Leap Motion can be integrated with the ROS system to control the robot. In the future, if our project is further developed such information would be significant.

A challenge in our proposed project is taking 2-D images from the TurtleBot's camera and transforming them into a 3D experience in Oculus. Lentin provides his readers with a link to his open source github project called ROS Cardboard which is based on ROS-Android APIs. This project transforms compressed images from a ROS PC and splits the view for the left and right eye to create a 3D feel for the VR headset. Although Lentin is using an Android application, much of the preliminary code and basic ideas will aid us in creating a successful framework for our project. Below is a figure (figure 2.1) of how the main components are communicating with one another to relay information. While the Gazebo simulator will be useful in preliminary testing, the long term goal is to acquire images from an actual TurtleBot. As aforementioned, ROS will be communicating directly with our Oculus VR headset instead of a VR android app.

Figure 2.1 [2] Communication between a ROS PC and Android Phone

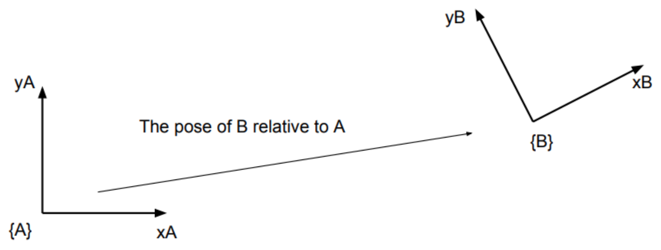


Mathias Thorstensen, a former master's graduate student at the University of OSLO in Robotics and Intelligent Systems, has done extensive work in visualization of robotic sensor data with augmented reality for his thesis in Spring 2017 [3]. In his paper, he discusses topics from data pipping to tools and software such as Unity which will be major components of our project. For instance, the position and orientation of our headset must be known in relation to the TurtleBot's environment. That is, its pose must be known. In section 2.2.2 of his paper, Thorstensen provides an algorithm that can be used to translate points from one coordinate system to another using transformation matrices. Although the provided algorithm is provided for 2D pose, it provides a basics of understanding which can then be expanded to account for 3D pose (figure 2.2). Furthermore, computer vision libraries, such as ArUco, use easily distinguishable markers in the environment to estimate the camera's pose (figure 2.3).

Figure 2.2 [3] 2D Pose Algorithm

$$\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & {}^{A}t_{Bx} \\ sin\theta & cos\theta & {}^{A}t_{By} \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.3 [3] The pose of coordinate system B relative to A



Thorstensen has opted to use the HTC vive virtual reality system that is equipped with the Intel RealSense f200 depth camera; Oculus uses a similar camera. He mentions that sensor data from an Intel RealSense depth camera can be viewed as a point cloud in rviz. The RealSense ROS package publishes the topics image_raw, camera_info, and points. Image_raw contains the raw depth images from the sensor, camera_info provides the camera matrix and distortion coefficients, and points houses the point cloud. While we will most likely not use Oculus' camera, understanding how Oculus takes in images and converts them to a 3D experience as well as tools that can be used for troubleshooting will be very helpful when dealing with converting ROS images.

After compressed images are sent to Oculus, they must be decompressed in order to access individual depth values. Thorstensen experienced difficulty when he decompressed the image from ROS using Unity's built-in image load function. He concludes that this issue was mostly likely due to the degree of compression and instead he opted to use the raw depth image topic. After he linked each depth reading with its corresponding pixel coordinate, transformation was achieved by translation of image coordinates into the normalized image plane, which corresponds to the camera's coordinate frame. Below is the matrix model he used as well as a depiction of transforming pixel coordinates and their depth measures into corresponding 3D points (figures 2.4, 2,5).

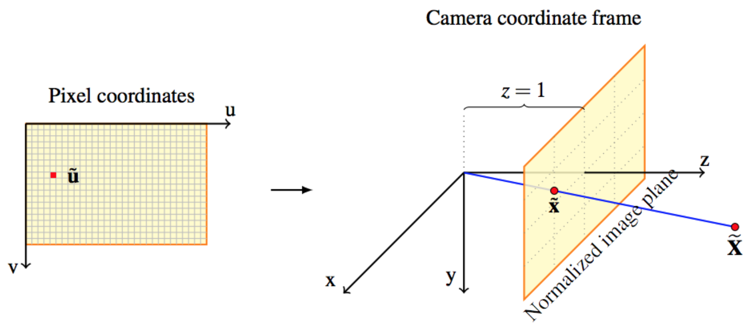
Figure 2.4 [3] Matrix Model - Image translation can be done by multiplying the image coordinates u with the inverse of the calibration matrix K, giving the corresponding point in the normalized image plane x.

$$\tilde{\mathbf{x}} = K^{-1}\tilde{\mathbf{u}}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{f_u} & 0 & -\frac{c_u}{f_u} \\ 0 & \frac{1}{f_v} & -\frac{c_v}{f_v} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{u}{f_u} - \frac{c_u}{f_u} \\ \frac{v}{f_v} - \frac{c_v}{f_v} \\ 1 \end{bmatrix}$$

Figure 2.5 [3] The transformation from pixel coordinates and a depth measure to the corresponding 3D point



Furthermore, in order to send sensor data to the computer responsible for the VR system, rosbridge_server was used allowing communication between ROS nodes on different computers using WebSocket for sending data. Thorstensen experienced issues such as ports being blocked on his university's local network. To solve this issues he used portforwarding. In additional to the technical details provided by this paper, Thorstensen discusses ways in which he evaluated his project. Factors such as learnability, efficiency, memorability, errors, and satisfaction will help us determine the success of our project while ensuring we are pursuing our goal in an effective manner. Thorstensen provides statistical methods to calculate such factors such as using null and alternative hypothesis.

In a York University study, robot and virtual environment middleware was researched and tested in efforts to find an optimal solution [1]. In the paper From ROS to Unity, it is discussed that while there exists middleware for robots and virtual reality systems respectively, these provided software are not easy to integrate. The authors purpose Unity as the best middleware and they discuss their approach to linking ROS and Unity. In their method, ROS and Unity require a precomputed map of the area of interest. Since TurtleBot requires a preprogrammed map for optimal function, this will not be a limitation for our project. This map is then augmented with real-time telemetry from the vehicle including local point clouds, and information from tilt and pitch sensors that are located on the robot. The tilt and pitch sensors would be an ambitious addition to our project. The paper goes on to discuss how ROS communicates with the external world. In ROS, messages are passed using an internal protocol based on TCP/IP sockets, however it is much simpler to use the rosbridge framework and WebSocket as a communication layer as it allows two way communication with an external source. Many libraries exist linking Unity and the WebSocket protocol. Rosbridge communicates messages to and from the external world in the form of yaml. Sometimes these yaml strings can be used directly by the external word, however, it may be convenient to use JSON, which creates instances of objects using Unity. With regards to the joy stick that is used to control the robot, input commands are converted into ROS messages of type sensor_msgsJoy.

3 Technical Details

The proposed system architecture is shown in the following figure:

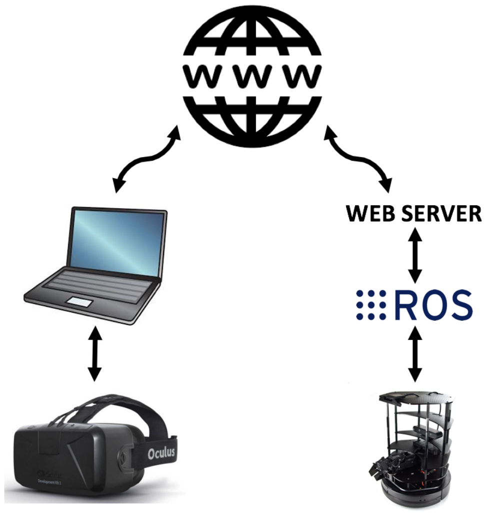


Figure 3.1 Overview of the ReVRSR system architecture

3.1 Communication between Turtlebot and Oculus SDK

A ROS instance is going to run locally on the TurtleBot. This instance is responsible for handling the translocation of the robot, collecting the 3D video data using Stereo Cameras, and finally responding to remote commands to the TurtleBot. For this project aims to be available through the Internet, a Server instance (which uses HTTP(S) protocol and has a public IP) is also going to be running on the TurtleBot. This Server is responsible for sending the relevant vision data from the TurtleBot to the Client, and for receiving remote commands and translating them to the ROS instance in the robot.

Understanding that many limitations might hinder the development of this system design, alternative plans have also been put through. First, if the the data transmitted is too much to transmit at a reasonable frame per seconds (fps) rate, the Web Server might switch from using a REST HTTP/HTTPS request architecture to using a different protocol. WebSockets is one candidate protocol, as it speeds up the transmittance of information by maintaining an open connection and removing Headers from each request. Additionally, the resolution of the 3D video might get reduced too. Finally, some of the data processing might be done on the TurtleBot's computer, so that the load on the Client's (viewer's) computer is reduced. All in all, we are planning to experiment with the setup and change its parameter so that it is optimized for 1) lowest network load 2) lowest load on the client's computer, and 3) most responsive and smooth experience.

3.2 Using Oculus SDK to Visualise the Data

The next step in the development process would be to translate and transform native ROS data structures into the ones supported by Oculus SDK natively. Fortunately, Oculus SDK and ROS are both implemented in C++ which means that it would be an easier process since the header files for the data structures are available in the source code that can be used.

We might also use OpenCV as our platform to perform image transformations. OpenCV is an open source Computer Vision library that is also implemented in C++ which aligns perfectly with the rest of our planned architecture. It's low-level highly optimized functions might be very useful in ensuring that the VR system is able to run smoothly in real time, without taxing too much processing power or network bandwidth.

We also intend to provide the user control over the robot through the Oculus handheld controllers. For this purpose, we would have to effectively do the reverse of the VR channeling described above and allow the Oculus to channel data back to the TurtleBot2. The extent and enhancement of that control is under our stretch goals.

References

[1] R. Codd-Downey, Mojiri P. Forooshani, A. Speers, M. H. Wang, and Jenkin. From ros to unity: Leveraging robot and virtual environment middleware for immersive teleoperation. In *Information and Automation (ICIA), 2014 IEEE International Conference on*, pages 932–936. IEEE, 2014.

[2] Joseph Lentin. *ROS Robotics Projects*. 2017.

[3] Mathias Ciarlo Thorstensen. Visualization of robotic sensor data with augmented reality. Master's thesis, York University, 2017.