Telepresence in a University Environment

Selena Groh, Serena Thoma, and Trung Truong

Abstract

This project created a telepresence system to be used in Halligan Hall at Tufts University. The system used an iPad displaying a Skype call to allow a remote user to communicate with others near the robot, along with the TurtleBot2 as a mobile base so that the remote user could exercise autonomy and drive their "presence" to any desired location within the building. A web-page user interface was created that allowed the remote user to control the robot with either manual commands, such as forward, backward, turn left, and turn right, or through autonomous map navigation, in which the user specifies a goal location on the map display, and the robot will then move to that location. The navigational capabilities of the robot were implemented using RobotWebTools libraries. Overall the system successfully implemented an efficient and usable telepresence system, but even so, there are many features that could added that would increase the range of uses for the system and simplify the user's experience even further.

1 Introduction

Imagine a child breaks her leg, missing months of the school year to recuperate. She comes back behind on her academics and isolated from her peers. Imagine a scientist needs to diagnose a critical problem in a system 1,000 miles away and all the flights are sold out. The system fails without the guidance of the expert. Imagine further that a young man is medically quarantined for years. He grows withdrawn from lack of interpersonal contact. These problems are unique, but each case could benefit from a system which makes remote attendance possible.

Telepresence systems are characterized by a video conferencing system attached to a mobile robotic base, allowing the user to pilot the robot remotely and see the environment surrounding the robot. The video conferencing system allows the user to interact with others through the robot, while the mobile base allows the the user a certain amount of autonomy in the surrounding environment. One main issue with telepresence is to figure out a piloting system that is easy and intuitive for the user, allows them to navigate between different locations quickly and efficiently, and prevents the robot from executing a command that may cause damage to the robot or its environment.

Telepresence robots can be used for a wide variety of scenarios, including healthcare, office spaces, elderly care, and education. For our project, we focused on the applications of telepresence in a university environment, more specifically, the computer science building at Tufts University. In Halligan Hall, a telepresence robot could be useful for both teachers and students who have to attend classes, meetings, or group projects but cannot be physically present. Additionally, a telepresence robot could provide interactive tours to prospective students who are interested in seeing the computer science facility but are unable to visit Tufts.

To create a telepresence system, we used a TurtleBot2 as a robotic base in combination with Skype on an Apple iPad to act as the video conferencing system for the robot. In order to control the movements of the TurtleBot remotely, we used the `rosbridge` package to create a server that allows omni-directional, asynchronous communication between different browsers (in this case the user's laptop and the TurtleBot). To implement teleoperation, or manual control of the TurtleBot's movements, we used the JavaScript library `keyboardtelopjs` written by RobotWebTools, which is able to detect keyboard inputs and publish forward, backward, and turn commands to the TurtleBot's velocity topic. To implement autonomous map-based navigation, we used the JavaScript library `nav2djs`, also written by RobotWebTools, which displays the subscribed `/map` and `/pose` topic to a browser and also publishes a goal when a location on a map is clicked. All these features were implemented together to create a web interface that allowed the user to control the robot both manually and autonomously, while also viewing live video and audio of the robot's surroundings.

2 Related Work

The paper "Teleoperation with Intelligent and Customizable Interfaces" focuses on the larger applications and challenges with teleoperation robots [1]. The study branched out to the larger application of using a teleoperational robot with a manipulator that allowed users to perform tasks with robot. The paper discusses the relationship between the robot and its operator, specifically the extent with which the robot is carrying out commands autonomously or with user control. Almost every task that a human carries out can be broken into sub-tasks, and the same is true for the action of a robot. Although the robot in our project will not be manipulating the environment, we will experiment with different levels of autonomous and user-controlled navigation to see how different types of commands affect both the accuracy with which the robot carries out the actions and the experience of the user when giving commands.

2.1 Navigation and Teleoperation

This project had the goal of implementing two different types of navigational control. The first goal was for the user to able to control the movements of the robot manually, and that pressing a single key or button on their computer the user could move the robot forward, move the robot backward, turn left, or turn right. The second goal was to implement autonomous movement through a map interface, that is, being able to specify a goal location on a map of the building and then allowing the robot to autonomously navigate from its current location to the goal location.

The paper "Building a Telepresence Robot based on an open source Robot Operating System and Android" documents how one can use ROS and Android package to control an iRobot Create robotic base [2]. This article includes both the software and hardware implementation of the telepresence robot, and we expect to explore and experiment at least one of the methods described in this article in our robotics model.

2.2 User Interface

To figure out the requirements of the user interface for our telepresence system, we investigated different resources. In one paper, the authors studied the navigation capabilities of a teleoperation robot in the context of healthcare professionals providing care to elderly patients in a home environment, allowing the patients to receive extended care more accessibly [3]. The majority of the article focuses on the interface for the user navigating the robot, and explores different types of navigation tools to assist the driver to maximize speed and efficiency while minimizing collisions. An initial user interface, which consisted of the video feed from the robot, a map of the surrounding area, a radar scan showing obstacles in the surrounding area, and a visualization tool of the joystick with which the user is operating the robot.

Figure 1: Telepresence Interface

This study had many interesting results that are applicable to our project. First, the study found that the users spent the most time by far looking at the portion of the screen that displayed the radar sensor data, which showed the user when there was an obstacle near the robot. The

users looked at this information about twice as much as they looked at the live video feed, even though the live video feed took up most of the screen. This comparison is important because it emphasizes that our interface should not only include the video from Skype, but we should also try to incorporate the laser scanner information from `rviz` into our user interface.

In addition, this paper proposes several different future iterations for a user interface that may improve its usability. Although these designs are very specific, the main decision for the next step was to choose between an egocentric design, meaning the user views information from the perspective of the robot, or an exocentric design, meaning that the user views information from a perspective separate from the robot, likely from above. Our design incorporates information from both of these perspectives, but the study overall suggests that focusing on the exocentric design may improve the usability of the interface.

3 Methodology

3.1 Problem Formulation

Our telepresence system needed to be able to effectively facilitate communication between a remote user and the robot's environment, including others in the robot's environment. Effective communication usually has a video or audio component. In addition, it is important that the remote user understand the robot's context in the environment, including its location and interactions with others in the environment.

In order to accomplish these goals, we knew we needed to augment the existing TurtleBot to accommodate a screen capable of receiving and transmitting audio and video between the two environments. We also needed to establish communication between the two environments and permit the remote user to control the robot's movement. We also knew that the remote user's experience needed to be as simple as possible in order to make the telepresence system widely accessible.

3.2 Technical Approach

The completed integrated telepresence system is shown in Figure 2.

Figure 2: Tufts Telepresence Robot (front and back view)

3.2.1 Hardware and Tablet

To create a telepresence robotic system, we mounted an Apple iPad, running a Skype video chat, onto a TurtleBot2[1] mobile base programmed with ROS.[2]

The iPad was mounted to the TurtleBot using several laser cut parts and two metal rods, allowing the iPad to display the the Skype call closer to a person's line of vision. The parts shown

[1] http://www.turtlebot.com/turtlebot2
[2] www.ros.org

in Figure 3 bolt directly onto the TurtleBot's lower platforms and hold two 24 inch pieces of 10mm T-slotted aluminum bars. The part on the left attaches to the TurtleBot's lower platform and the ends of the T-slotted bars are bolted to the lower two holes; the part on the right is bolted to the TurtleBot's upper platform and secures the bars as they extend through the two square holes.

The parts in Figure 4 show the frame that holds the iPad at the top of the metal bars. The part on the left goes in front of the iPad, allowing full access to the screen, the home button, and the front camera. The part on the right goes behind the iPad, and is attached to the T-slotted bars with two steel L-brackets, allowing the angle at which the iPad is mounted to be adjusted according to the user's preference. These two parts are bolted together on either side of the iPad in such a way that the iPad can be removed easily, the volume and lock buttons on the sides can be accessed easily, and the iPad can be plugged in if it needs to be charged, all without damaging the iPad itself.

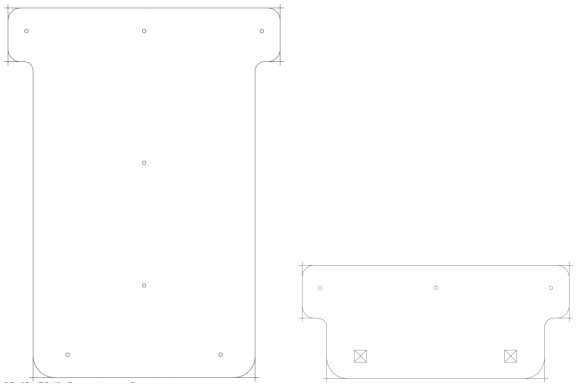

Figure 3: Laser-cut parts base mounts

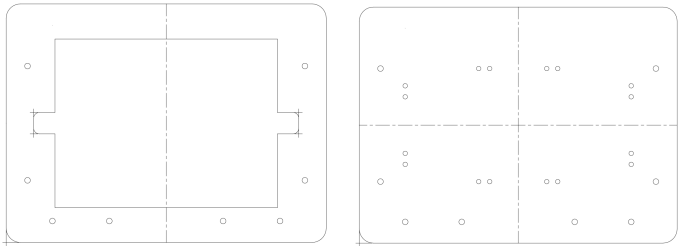

Figure 4: Laser-cut parts iPad mounts

3.2.2 Communication Protocol

Communication between the user's computer and the TurtleBot is essential to allow remote operation. In order to prevent lag and establish communication, we used the `rosbridge`[3] package, which allows omni-directional, asynchronous communication between different browsers, visualized in Figure 5. Running the command

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

creates a `rosbridge` server on the laptop controlling the TurtleBot. The websocket server can be accessed on any modern browser with the appropriate IPv4 address. The browser on the client's side includes `roslibjs`, a JavaScript library capable of publishing and subscribing to ROS topics.

3.2.3 Teleoperation

After a secured connection is set up, the client must be able to publish to the TurtleBot's velocity topic. `Keyboardtelopjs`[4] is a pre-written JavaScript library written by RobotWebTools that

[3] http://wiki.ros.org/rosbridge_suite
[4] https://github.com/GT-RAIL/keyboardteleopjs

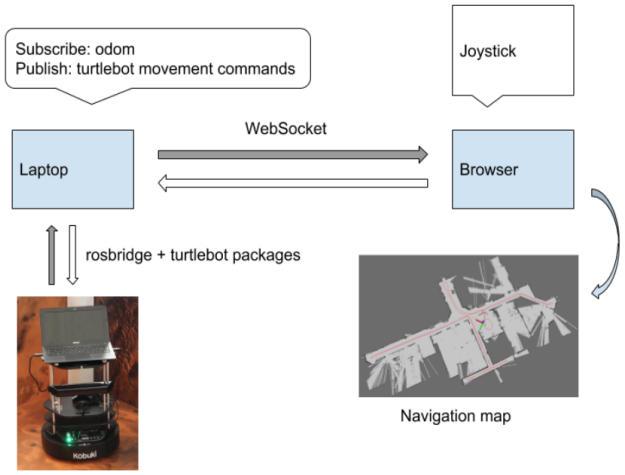

Figure 5: Graphical representation of communication protocol

detects keyboard inputs and publishes to the `cmd_vel` topic accordingly. With this library, the robot will move forward by pressing the `w` key, move backward by pressing the `s` arrow key, turn left by pressing the `a` key, and turn right by pressing the `d` key. This feature is particularly useful for fine-grain position adjustments; however, it is tedious for extended operation and does not provide obstacle-avoidance procedures. For these reasons, we implemented autonomous navigation functionality.

3.2.4 Map-based Autonomous Navigation

In order to achieve autonomous maneuver function, we used the the JavaScript library `nav2djs`[5], written by RobotWebTools. `Nav2djs` displays the subscribed `/map` and `/pose` topic to the browser and publishes a goal when a location on a map is clicked. `Nav2djs` is based on `roslibjs`, which allows internet communication between different ROS environments, and `EaselJS`, which implements an interactive 2D canvas in HTML5. By publishing a goal, we were able to take advantage of the TurtleBot's obstacle-avoidance capabilities and allow the robot to plan the most effective and efficient route to the goal instead of relying on manual human operation.

3.2.5 User Interface

All of the features described above were included in a web page that acts as the user interface for the remote user. The web page, shown in Figure 6, displays instructions for the user to connect to the TurtleBot, a bar to enter the IP address and port number of the TurtleBot to form the connection, the connection and battery status of the TurtleBot, and a map that shows both the current position of the robot and goal position of the robot.

The instructions describe what actions must be taken on the TurtleBot and on the webpage in order to connect to and control the TurtleBot effectively. The connection bar allows the user to connect to the TurtleBot of their choice. The connection and battery status provide the remote user with important information about the robot, which could be extended by subscribing to additional ROS topics beyond battery status. The map allows the remote user to locate the robot in the environment and click on the map to change the goal position. We ensured that the map was prominent as prior research has demonstrated the importance of exocentric design features in telepresence, as mentioned previously [3].

Figure 6 also displays one possible configuration for viewing the Skype call with the TurtleBot's iPad alongside the web page interface. The TurtleBot has its own Skype account (tuftstelepresence), as well as its own Google Hangouts and FaceTime accounts. The Skype call was not embedded directly into the webpage as few major video calling services allow that feature. However, it is simple for the remote user to set up their web browser application and their Skype application so that they can view both simultaneously on their screen, so this does not pose a great limitation.

[5] http://wiki.ros.org/nav2djs

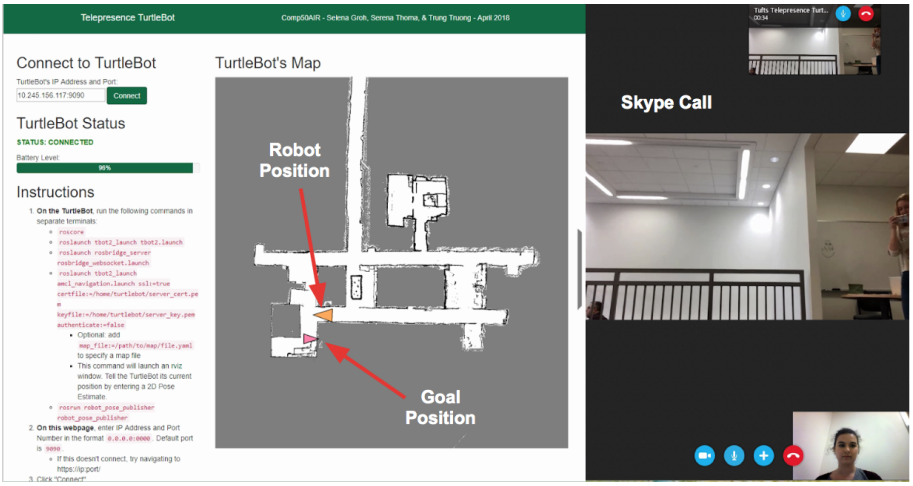

Figure 6: User Interface (browser view)

4 Experiments and Results

4.1 Example Run

For our example run, we completed a sequence of common tasks consisting of directing the robot through the web page interface to move from one distinct location to another. To fully test the usability of the entire system, the test user also needed to successfully connect to the TurtleBot using the instructions on the web page interface.

The full procedure of the user test included the following:

1. Remote user initiates a Skype call to the robot's iPad, which is answered by a different user near the TurtleBot
2. User next to the TurtleBot runs the following commands on the TurtleBot:

```
roscore
roslaunch tbot2_launch tbot2.launch
roslaunch rosbridge_server rosbridge_websocket.launch
roslaunch tbot2_launch amcl_navigation.launch
  ssl:=true
  certfile:=/home/turtlebot/server_cert.pem
  keyfile:=/home/turtlebot/server_key.pem
  authenticate:=false
```

3. Optional: add `map_file:=/path/to/map/file.yaml` to the last command to specify a map file.
4. The last command command will launch an `rviz` window. Tell the TurtleBot its current position by entering a 2D Pose Estimate.
5. Run the command `rosrun robot_pose_publisher robot_pose_publisher` on the TurtleBot.
6. Remote user connects to the TurtleBot using its IP address and port number
7. Remote user enters a goal location for the robot using the map interface
8. TurtleBot navigates from initial location to goal location
9. Remote user checks that robot has arrived at the correct goal location using Skype call
10. Remote user tests manual controls by testing all arrow keys, monitoring resulting motion through Skype call

By following all of these steps, all of the functions we implemented in the project were tested, and could be assessed by a third party observing the robot.

4.2 Quantitative Results

The nature of our project yields little quantitative results, as our goal was to maximize the usability and efficiency of the system. We ran the test described in the previous section three times, and all three times all the steps were fully completed and the robot was able to reach the final destination successfully. A full test run can be viewed on YouTube.[6]

4.3 Discussion

Although the tests were completed with a 100% success rate, there were some interesting findings. The most notable issue that affected the performance of the robot was human interference. While the robot was completing its movement from its starting location to the goal location, we tried different ways of impeding the robot's navigation, including introducing an obstacle in the robot's path, varying the final orientation of the robot (meaning the direction the robot was facing), and physically picking up and moving the robot. Introducing obstacles and changing the robot's direction ended up being minor inconveniences to the robot; the robot easily implemented obstacle avoidance to navigate around objects it detected in its original path, and the user could easily use teleoperation to turn the robot to face the desired direction. Picking up and moving the robot proved to be a more difficult challenge to overcome, as it required the robot and the user to recalibrate the robot's current position. Luckily, it is reasonable to assume that people will not want to purposely interfere with the robot, especially when the remote user can clearly communicate with people in Halligan through the Skype video call.

5 Conclusion and Future Work

Overall, our group was able to successfully implement a telepresence robotic system in Halligan. A user was able to remotely connect to our system, control the robot manually through key teleoperation, and control the robot's location through map commands, all while sharing video and audio of the two environments. Although this system was fully functional and usable, there are many features that could be added to the system to increase the efficiency of the robot and widen the scope of possible uses for the robot. If future work was to be done on this project, here are several areas that could be expanded upon:

- Add up and down tilting movement to the tablet camera to ensure full visibility of surrounding environment for user. Currently, the angle of the iPad can be controlled by loosening the bolt attaching the L-bracket to the bar, but motorizing this function would give direct control of the angle to the remote user without needing additional users or tools near the robot.
- Include a dynamic map that shows surrounding obstacles. Currently, the map displayed on the web page is static other than the two arrows showing the robot's current position and the goal position. If the map could additionally display shapes the robot detects from its laser scan (the costmap), the remote user would be alerted of obstacles around the robot that are not included in the map itself, such as chairs, tables, trashcans, and people. This could be particularly useful as the robot's current camera angle cannot provide a full view of the surroundings, especially obstacles close to the robot and low to the ground.
- Embed Skype video conferencing into the web interface. This function proved to be too time consuming to fit within the scope of our project, but adding the two windows together would greatly simplify the system for the user. One caveat to this feature would be that the user is limited in the type of video conferencing they must use, as the user currently could also use Google Hangouts or Facetime instead of Skype.
- Add semantic mapping to the Halligan map. This could include adding room numbers or professors' names to their offices, which would increase the user's understanding of the map and interface, as well as allow the remote user to simply specify a semantic location rather than an exact point on the map.

[6] https://www.youtube.com/watch?v=Z8beKiPEZjg

- Expand the map to include other locations on campus. Other locations around campus could also benefit from a telepresence system. Adding this feature would require using the TurtleBot to map other spaces around campus to create a library of map files for other locations, which would then be published to ROS's `/map` topic and loaded into the web interface upon connection with the TurtleBot.

- Simplify and streamline startup commands on the web interface. The system currently requires five commands to be run on the TurtleBot, which may be confusing to those unfamiliar with ROS. Consolidating these commands would simplify the software startup on the TurtleBot and make the system more accessible to the public.

The goal of this project was to create a system that is as easy to use as possible, and implementing these features would bring us one step closer to creating an inclusive and collaborative space in Halligan for all students, even those who are new to computer science.

Our work can be found on GitHub,[7] and the web page interface is also live.[8] A full presentation of our work can also be found here.[9]

References

[1] Dragan, A. D., Srinivasa, S. S., Lee, K. C. (2013). Teleoperation with Intelligent and Customizable Interfaces. *Journal of Human-Robot Interaction, 02(02), 33-57.* doi:10.5898/JHRI.2.2.Dragan

[2] Do, Ha M. Mouser, Craig J. Sheng, Weihua. (2012). Building a Telepresence Robot Based on an open source Robot Operating System and Android *Theoretical and Applied Computer Science* (TACS 12)

[3] F. Michaud, P. Boissy, D. Labonté, S. Brière, K. Perreault, H. Corriveau, A. Grant, M. Lauria, R. Cloutier, M.-A. Roux, D. Iannuzzi, M.-P. Royer, F. Ferland, F. Pomerleau, D. Létourneau, Exploratory design and evaluation of a homecare teleassistive mobile robotic system, Mechatronics, Volume 20, Issue 7, 2010, Pages 751-766, ISSN 0957-4158, https://doi.org/10.1016/j.mechatronics.2010.01.010.

[7] https://github.com/selena-groh/tuftstelepresence
[8] https://selena-groh.github.io/tuftstelepresence/
[9] https://tinyurl.com/tuftstelepresencepresentation