# Physical Privacy Interfaces for Remote Presence Systems

**Authors**

**Student**
Suzanne Dazo
Berea College
Berea, KY, USA
suzanne_dazo14@berea.edu

**Mentor**
Dr. Cindy Grimm
Oregon State University
Corvallis, OR, USA
grimmc@onid.oregonstate.edu

## ABSTRACT

Research in the Applied Robotics Lab at Oregon State University focused on privacy interfaces for robots navigated by a remote operator in a user's home. Privacy expectations and its components are defined, and this definition is used to develop three applications for both a local user and a remote operator using ROS. The first application, Remote Nav, is a user interface used to navigate a Remote Presence System. The second is Privacy Zones, which the local user can use to specify areas that should remain private or public. Both applications are functional but are pending testing in a user study or similar testing environment.

## INTRODUCTION

Over the summer, I worked in the Applied Robotics lab at Oregon State University under the mentorship of Dr. Cindy Grimm. Research in the lab focused mainly on human-robot interaction under Dr. Bill Smart. The goal of the lab is to improve the interactions between robots and people and determining how they can be useful.

Remote Presence Systems (RPS) are systems that allow a remote operator to be virtually present in another location. One example would be the Personal Roving Presence (PRoP) Project [4], which is a mobile robot that navigates the home of a local user. It has the ability to interact with the people that it comes across. Expanding upon this, an RPS could also have to ability to interact with objects in the environment.

With the introduction of these RPS, there are also privacy concerns that are raised. A remote operator can point the camera in any direction, and the RPS can be present anywhere in the space that it can physically access. The purpose of this interface project was to develop a solution for specifying privacy in the space of the local user, as well as upholding that privacy in the context of what the remote operator can do.

Overall, our goal will be to meet the privacy expectations of the local user, and holding up that standard when the remote operator controls the RPS. This project was the foundation for further work with privacy interfaces which may include future user studies and expansion of the original capabilities of the applications.

## BACKGROUND

To best understand how we can uphold the privacy expectations of the local user, we should define what those privacy expectations should be. This involves the *privacy type*, or what the restrictions are on the RPS, and the *privacy context* which are the conditions under which the *privacy type* is enforced. This should be applicable with a wide variety of tasks. In developing these applications, various tasks were considered in order to meet these requirements. The privacy type that was focused on in my research was *physical privacy*, which deals with where the RPS can go and what it can interact with. In the context of this project, the goal is limiting where it can go. This falls under two categories:

**Can't Enter Area** – The RPS cannot enter a "private" space, and would navigate around such areas. The RPS would refuse requests of navigation goals into private areas.

**Can't Leave Area –** The RPS cannot leave a "public" space, and would not navigate outside of this space.

Ideally, the result of the remote operator violating these privacy types would be removing control from the remote operator and autonomously navigating until the RPS is in a valid location.

There are various privacy contexts such as temporal time constraints, and the presence of a person or object in the room. However, there are only two privacy contexts that concern us for this project.

**Spatial-** The privacy type is enforced when the RPS in in a particular room or a defined area.

**Remote Operator-** The privacy type is enforced depending on who the remote operator is. This can vary based on the identity of the operator or the role/task of the operator. For example, an RPS controlled by a plumber expected to work in the bathroom would have different constraints compared to an RPS controlled by a home inspector (who would probably have the freedom to explore the entire home).

## APPLICATION

The development of this project involved two phases due to the fact that there are two types of users involved in physical privacy. The Remote Nav package was developed for the remote operator of the RPS. The Privacy Zones application is intended for the local user. Both of these programs were developed using a combination of Python, PyQt, and various packages included in ROS[11, 8, 5] and can be found on the Oregon State Robotics Github[1].

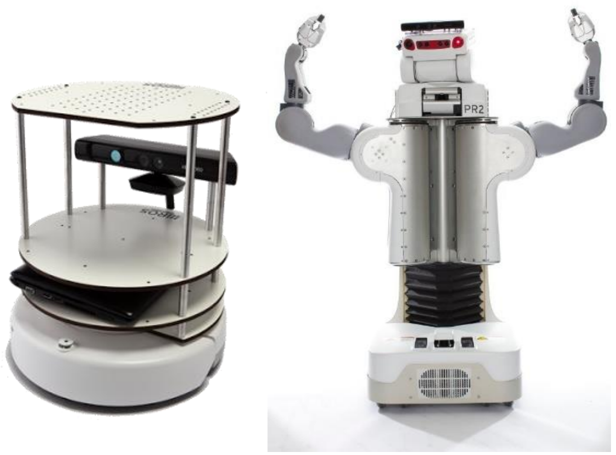


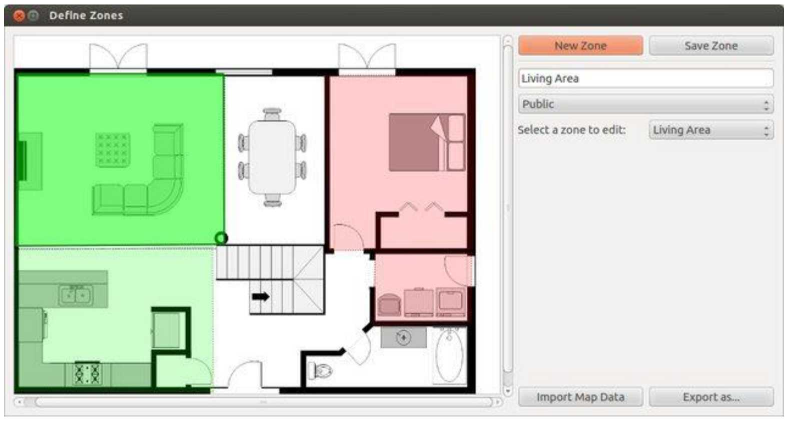
*Figure 1: A TurtleBot(left) and PR2(right)*

### *Framework*

The Robot Operating System, or ROS, is used across multiple platforms to provide functionality to various robots. It is open source, and allows for the publishing of developed packages for use by other developers. This allows for rapid use of new research to enhance the development of future robot applications. As of this writing, the current version of ROS is Hydro, but for compatibility reasons, ROS Groovy was used for development.

All of our applications should work in any version of Linux that can also support ROS. We have worked with prototyping our applications on two robots, the PR2 and the TurtleBot (*Figure 1)*. Some major differences between the PR2 and the TurtleBot are that the TurtleBot runs on a single base and can move forward, backwards, and rotate left and right. It has a single Kinect several inches above the base which is where it obtains depth and visual information. The PR2 is much more robust with a base (*Figure 1, the bottom of the robot)* than has wheels that can move forward, backwards, left, and right as well as rotate. It also has a series of sensors on the head, which includes a Kinect and separate stereoscopic cameras. The head can move separately from the base and can look left, right, up, and down. A challenge here is to create an interface that has the same controls between both robots, but also allows both to have proper mapping between the UI controls and the real-world movements.

### *Specifying Physical Privacy with Privacy Zones*



Living Area
Public
Select a zone to edit: Living Area

*Figure 2: The Privacy Zones Interface with the floor plan for an example home.*

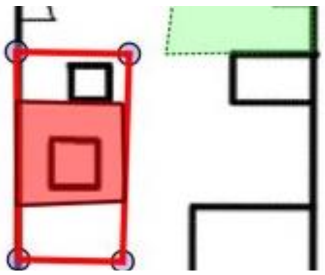
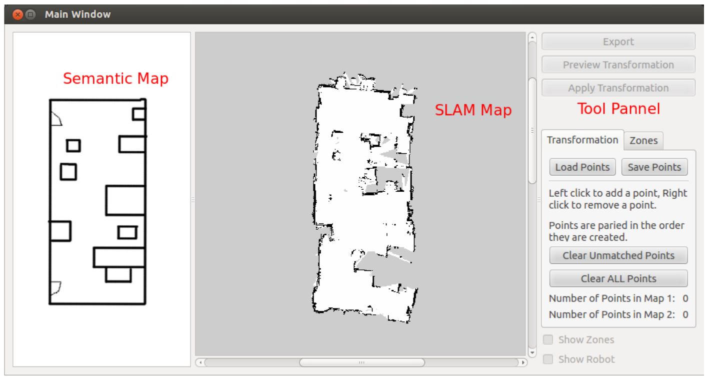


*Figure 3: Editing points*

The Privacy Zones application allows the local user to designate what areas in the home are public, private, or neutral. By clicking on this semantic map, points are drawn at the point where the user has clicked. When enough points have been specified, the application will automatically lasso around the area to show the user what they have selected. This is similar to the behavior of the Polygonal lasso tool in Adobe Photoshop[10]. These individual points can be dragged to edit the selected area and further fine-tune a selection (*Figure 3*). When the user is done specifying an area, "Save Zone" converts those points to a zone and indicates its current privacy setting:

**Green** – The RPS will treat this area as public.
**Red** – The RPS will treat this area as private.
**Gray** – The RPS will treat this area as neutral, and it is only for labelling purposes until it is edited to become either public or private.

Due to the fact that there may be multiple remote operators with different tasks, the interface allows for the saving of these collection of zones. For example, *Figure 2* demonstrates the Privacy Zones interface with the bedroom and bathroom as private zones. A new file could be created with a different privacy setting for a different remote operator. If the user would like to change an already existing map file, clicking "Import Map Data" would load a save file for further editing or review.



Left click to add a point, Right click to remove a point.

Points are paried in the order they are created.

Number of Points in Map 1: 0
Number of Points in Map 2: 0

*Figure 4: Map Registration finds the relation between the semantic map data and the real-world coordinates of the SLAM map.*

An interesting problem is that the map that a user might understand is drastically different from the SLAM map a robot uses to navigate a space [6, 7]. SLAM maps, due to being a compilation of laser scan data, may be crooked, have jagged edges, and do not offer any identification of what rooms or furniture are in an area. Because of this, we also have decided to work with semantic maps that a user can understand.

The intermediate step between the zones created by the local user with Privacy Zones and the navigation involves converting the coordinates of the image from the semantic map used into real-world coordinates for use by the RPS. Map Registration [1] (*Figure 4*) allows any user with knowledge of the space (this does not have to be the remote operator or the local user) to convert the points of the semantic map into the SLAM map and saves the converted points to a file.

## *Navigation of the RPS using Remote Nav*

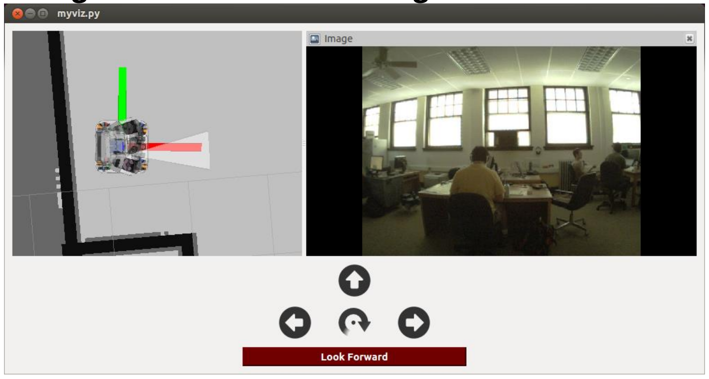


*Figure 5 The Remote Nav Interface with the PR2*

*Figure 5* demonstrates the current iteration of the Remote Nav interface which allows a robot to navigate a room. There are two input feeds. The left feed is the semantic map with a robot model placed on the map. This map can be zoomed in and rotated to provide better understanding for the user when navigating. A cone of vision is also projected from the robot model, demonstrating the direction the head is currently facing. On the right side is a camera feed from the Kinect camera to allow the navigator of the RPS to see the environment in order to complete a task.

To prevent the robot from backing up into something it cannot see or detect with its laser scanners, the robot is only allowed to move along a pre-programmed track as of now. Clicking the up arrow will move the robot in the direction it is facing along this track. Clicking the rotate button will cause the robot to turn around on the track so it may return to a previous location.

There are several key differences between a TurtleBot and PR2 with Remote Nav. First of all, a TurtleBot's Kinect camera is attached to the base, so all movements will also move the camera. The left and right buttons in turn would cause the TurtleBot to rotate in place, while

[1] This was mainly developed by Penn Biggs, another REU student, with the UI developed by me. More information can be found on the OSU Robotics Wiki[1]

on a PR2 it will cause the robot to look left and right with the head, while keeping the base in the same position. The reason for this is that a TurtleBot is significantly smaller than a PR2 and rotating in place is less likely to cause an accidental collision than if the PR2 were to perform the same action.

The “Look Forward” button acts as a reset position. For the PR2, it will reset the head to face forward, and for the TurtleBot it will rotate it to be parallel with the pre-programmed track that it will follow.

The purpose of this interface was to allow for easy navigation of a PR2 or TurtleBot that did not involve using a joystick or keyboard control, provided enough information for navigation (a map and visual data) and was safe for the robot. There should be little room for user error as far as navigating the robot into a dangerous or damaging situation, and it should automatically avoid obstacles such as furniture.

## CONCLUSION AND FUTURE WORK

### *Expanding Functionality*

Although the Privacy Zones and Remote Nav applications are meant for the local user and remote navigator respectively, they do not yet form a cohesive unit. As of this writing, Remote Nav does not yet pull the zone data generated by Privacy Zones, and this will be a major next step to ensure that navigation can uphold the privacy expectations of the local user. This will involve manipulating costmap[9] data so that the robot will not choose to navigate into an area that does not conform to a privacy expectation. Special care will need to be taken that the remote operator cannot manipulate these zones to help preserve the privacy of the local user.

As far as privacy expectations, types, and context goes, the Privacy Zones interface can be expanded to allow for more privacy contexts. This could involve adding a time parameter (ex: a public zone becomes private after 5pm) or other such rules.

### *User Studies*

Pilot testing of these interfaces and testing of how users interact with these interfaces could be done. Does upholding a privacy expectation reduce task performance of the RPS? How does a local user trust that the privacy that they expect is being upheld? These are a few of the questions that can be answered with future research and user studies.

### *Beyond Physical Privacy*

Privacy types extend beyond simply where an RPS can go. The next steps, especially with the PR2, can involve what the RPS can interact with (move/touch). In fact, privacy types expand beyond even physical privacy. Some work in Dr. Smart’s lab involves video filters to also handle visual privacy [2, 3]. Combining these filters with the physical privacy work seen here can eventually create an all-encompassing privacy interface that we will be able to test with a RPS.

References

[1] Privacy Interfaces, (2014), GitHub repository, https://github.com/OSUrobotics/Privacy-GUI

[2] Daniel A. Lazewatsky and William D. Smart. An inexpensive robot platform for teleoperation and experimentation. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 1211–1216, 2011.

[3] Daniel A. Lazewatsky, Bogumil Giertler, Martha Witick, Leah Perlmutter, Bruce A. Maxwell, and William D. Smart. Context-aware video compression for mobile robots. In Proceedings of the IEEE/RSJ International Conference on Robots and Systems (IROS 2011), pages 4115–4120, San Francisco, CA, 2011.

[4] Eric Paulos and John Canny. Social tele-embodiment: Understanding presence. Autonomous Robots, 11:87–95, 2001.

[5] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: An open-source robot operating system. In ICRA 2009 Workshop on Open Source Software in Robotics, Kobe, Japan, 2009.

[6] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. Artificial Intelligence, 128(1-2):99–141, 2000.

[7] ROS Navigation Package, (2014), GitHub repository, https://github.com/ros-planning/navigation

[8] Qt Project. (n.d.). Retrieved August 20, 2014, http://qt-project.org/

[9] Costmap 2D Package Summary - ROS Wiki. (2014). Retrieved August 20, 2014, http://wiki.ros.org/costmap_2d

[10] Photoshop Help | Selecting with the lasso tools. (n.d.). Retrieved August 20, 2014, from http://helpx.adobe.com/photoshop/using/selecting-lasso tools.html#select_with_the_polygonal_lasso_tool

[11] Python Software Foundation. (n.d.). Python. Retrieved August 20, 2014, from https://www.python.org/