Skype: a communications framework for robotics

Peter Corke, Kyran Findlater and Elizabeth Murphy

***Abstract*— This paper describes an architecture for robotic telepresence and teleoperation based on the well known tools ROS and Skype. We discuss how Skype can be used as a framework for robotic communication and can be integrated into a ROS/Linux framework to allow a remote user to not only interact with people near the robot, but to view maps, sensory data, robot pose and to issue commands to the robot's navigation stack. This allows the remote user to exploit the robot's autonomy, providing a much more convenient navigation interface than simple remote joysticking.**

I. Introduction

Robotic telepresence and teleoperation is an increasingly important application for robots to meet everyday needs in security, business meetings, health and aged care [1], and numerous other domains. It has been shown that the ability for a remote user to move in the local environment enhances the quality of the interaction for both parties [2]. Already there are a number of products on the market aimed at these applications.

To date most work in teleoperation has been focussed on applications such as bilateral control of robot manipulators for underwater, space and even health applications where the challenges are around motion fidelity, haptic feedback and dealing with communications delay. Mobile robots are increasingly common and have relatively simple motion and control requirements yet creating a mobile telepresence robot remains challenging.

Mobile robot telepresence requires both *mobile robot control* and *telepresence* capability. In a short space of time ROS [3] has become ubiquitous for *mobile robot control* by providing "out of the box" sensor interfaces and robust navigation capability. The dominant *telepresence* tool is Skype, a free tele- and video-conferencing software package that also provides features such as chat, file exchange and desktop sharing. In this paper we describe how these two very different technologies can be integrated to create powerful telepresence robot systems. Skype's ubiquity on mobile devices makes it a sufficient tool to interact with a remote user via a robot from anywhere on the planet.

While robot navigation, remote control and video conferencing are all well known there is a gap in their integration. Remote control using basic motion commands like forward and turn is challenging in practice and not very efficient. Part of the problem is that a monocular image with limited field-of-view gives a limited sense of the immediate environment, particularly if that environment is unfamiliar or very busy.

With the CyPhy Lab, School of Electrical Engineering and Computer Science, Queensland University of Technology, Australia firstname.lastname@qut.edu.au

Fig. 1. The Guiabot showing Skype interface on the top screen.

A more important problem comes from lack of situational awareness in an unfamiliar environment — it is easy to get lost [4]. A map, and our current position with respect to the map, makes remote operation of the robot much more productive. ROS provides a robot with an ability to localize and navigate with respect to a map, so the challenge for telerobotics is to share the robot's state with the remote user, and to accept commands from the user that are referenced to the map. Integrating Skype with external software is possible but this raises a technological issue since Skype's best known integration tools are for Windows whereas most robot platforms using ROS run Linux.

This paper describes an architecture that integrates Skype and ROS to provide a powerful means for remote operation of robots. We discuss two systems: text chat control using a standard Skype client; and map-based control using the Skype development environment (Skypekit). The architecture

has been demonstrated on the Adept Guiabot shown in Figure 1 and controlled by remote users in the same building, the same city or on a different continent.

The remainder of this section provides a discussion of prior work in telepresence robotics and an introduction to the aspects of Skype relevant to teleoperation and telepresence. Section II describes the text chat based architecture and Section III describes the map-based architecture. Section IV covers the important topics of security and safety, then Section V reflects on the lessons learnt, portability issues and discusses future work, and finally Section VI presents conclusions. Our source code is available from `http://tiny.cc/cyphy/software`.

A. Telepresence robots

There are a large number of mobile robot telepresence systems spanning a significant price range. At the low end are hobbyist or DIY projects which can be built for less than $500. At the high end are commercially produced telepresence robots such as the iRobot AVA [5] (not for sale at this time) or the Anybots QB [6] which currently sells for approximately $10,000. It comprises a mobile base with extendible 'neck' for a head which has a small LCD screen and two cameras — one for telepresence and human interaction and another to assist with driving. The Anybots system does not use Skype, but three of the hobby robots do. The sophistication and complexity of these systems varies considerably but the common feature is basic teleoperation commands by which the remote user can move the robot forward, reverse it, and turn left or right. Most commonly these commands are keyboard button presses.

The hobbyist or DIY telepresence robots are interesting examples of what can be done. Johny Lee's low-cost robot [7], [8] utilises a netbook mounted on top of an iRobot Create platform. The netbook communicates with the base via a serial link to send drive and docking commands and also to monitor battery and sensor state. A UI on the netbook allows a local user to control the robot and also listens for drive commands over the internet. This command channel is in addition to the Skype channel for telepresence, resulting in increased complexity, lower robustness and potential security problems. The remote user needs to run a separate network sender program to connect to the robot.

Sparky, and the newer Sparky Jr. projects [9] are DIY open source projects utilising netbooks or cut down Mac mini computers and open source microcontroller boards such as Make or Arduino to run the motors. The Sparky project has been running for over 15 years, starting in 1994, and has a fairly well established open source code base. Skype runs on board and they use a 'Skype plug-in' which listens to Skype, parses incoming text chat commands from the home user, and sends those along to the motor controller software which is linked to the Make/Arduino board. The remote user needs nothing more than the standard Skype client to call and control the robot.

David Schneider [10] used a similar approach to Johnny Lee. He fabricated his own moving base which is controlled by an Arduino microcontroller and which also carries a the laptop/netbook. He used used C# code by Hari Wiguna [11] to make use of Skype's Skype4COM windows-only desktop API [6] and modified it to send commands to a pan/tilt camera and also to the mobile robot base. His program parses text chat messages from Skype and sends single character commands over a serial link to the motor controller.

B. Skype 101

Most people are familiar with the Skype application for teleconferencing. Skype is a voice over IP (VOIP) tool that was created by Niklas Zennström and Janus Friis and first released in 2003 as a Windows application. MacOS support came in 2006 and Linux in 2008, and it is also supported a wide range of mobile devices (iOS, Android and Blackberry). The Skype protocols are proprietary but it uses open video codecs VP7 and VP8 from On2 Technologies (now owned by Google) and H264 for HD video. The main audio codec is the Skype-developed SILK. Skype has nearly 700 million users and since 2011 is owned by Microsoft.

Skype clients use peer-to-peer[1] communications as a distributed database lookup service for call initiation. Supernodes form an overlay network that help connect all Skype clients together and also to the Skype authentication server. Any Skype client outside a firewall will serve as a supernode (this can be disabled in modern versions of Skype), and other Skype owned supernodes are dotted around the planet. The Skype login process involves the client authenticating their user name and password with the Skype login server which holds all user names and passwords.

Skype supports interaction with external programs running on the same computer but this is complex, platform specific and has varied considerably over time. The oldest and most mature interface is for Windows: Skype4COM is an ActiveX component that allows control of Skype within an ActiveX environment. An older open-source Python interface called Skype4Py is somewhat buggy and the project is no longer active.

The desktop API (previously called the Public API) provides a string based interface to control the local Skype client [12]. Functionality includes call initiation, contact list lookup, SMS, chat, file transfer. For Linux the API uses DBUS which is an open message bus system that allows applications to communicate with one another, and it underlies the KDE desktop system. DBUS has bindings for many languages including C++ and Python thus allowing programs written in those languages to control a Skype client. For MacOS the API is a Carbon or Cocoa framework and or AppleScript.

If instead of remote controlling a Skype client we wish to build Skype capability into our own application we need to follow a different path. SkypeKit comprises the so-called *run time* which is a "headless" Skype client, a background process that communicates with the Skype network and to

[1]The name is derived from Sky peer-to-peer. The peer-to-peer protocols were originally based on the Kazaa software created by the same developers.

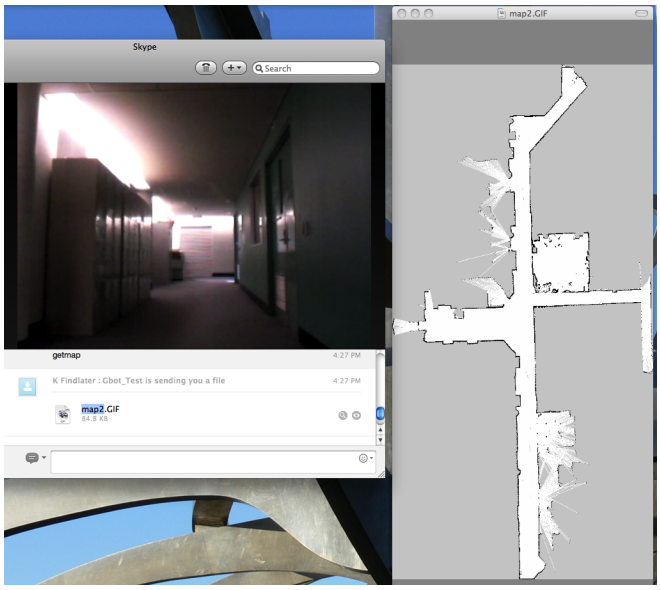

Fig. 2. Remote user view for the chat-based interface. A standard Skype client is used and a map can be downloaded from the ROS navstack and displayed.

which various user-written applications connect via an API [13]. Language bindings exist for C++, Python and Java[2] and this is a good fit with ROS where the most mature APIs are also C++ and Python. Skypekit has no GUI support but does perform audio input and output and video input via the operating system.

For robotics applications the most valuable capability of Skype is the so-called App2App (application to application) functionality. This is effectively a named communications channel between processes running on the computers at either end of the Skype connection. Processes can exchange strings or binary datagrams (upto 32kbyte) and these are multiplexed through the Skype connection. Each process needs to register the name of the App2App stream, which is an agreed string, and receives a handle. The sender writes strings or binary datagrams to the handle and at the receiver a callback is invoked with the data and the identity of the App2App channel. Importantly, new App2App streams can be created dynamically at run time and the number of streams is effectively unlimited. Message delivery is not guaranteed and if bandwidth is constrained then message delivery rate will suffer.

II. Chat-based navigation

The chat-based navigation system requires only that the remote user has a standard Skype client, on a desktop or mobile platform. The user calls the robot, which has a Skype username and enables video. The user then sees a video stream from the robot's navigation or human interaction camera, and in the latter case can have a voice and video interaction with people near the robot. This much is standard "out of the box" Skype.

[2]The Java binding is embryonic at the time of writing.

goto N	Move to the location called N
getmap	Send the map using file transfer protocol
where?	Report robot's location
fwd X	Move forward X meters
turn X	Turn X degrees, positive is to the right
left	Turn 90 degrees to the left
right	Turn 90 degrees to the right
navcam	Select the robot's navigation camera
usercam	Select the robot's user interaction camera

TABLE I

Subset of commands accepted by the robot via the Skype text chat interface.

In addition the user can send text chat commands, listed in Table I, which are interpreted by a robot-end server process. The robot responds, via a text chat message, when the command is complete.

The overall architecture is shown in Figure 3. The robot-side application is written in Python and incorporate both the Skypekit and ROS API and acts as a relay between the two. The program is effectively a daemon or server and once connected to ROS and logged into Skype it waits for connections and commands from the user. The robot must be running the Skypekit runtime, ROS Core and the Navigation stack (with a previously generated map from the Gmapping ROS node of the area in which the robot will be operating).

The SkypeKit API uses callbacks to notify the program about changes in status such as incoming calls or chat messages. For example if the user types "fwd 5" then the robot-side server invokes a callback which parses the command and dispatches an appropriate method to handle it. The method sends the feedback "Forward command received" to the user, and sends a goal to the navigation stack through the ROS topic /goal which causes the robot to move forward five metres. While moving, the program monitors the current pose and confirms that the movement has finished, by sending the feedback "finished moving" which indicates to the user that the robot has completed the movement. This is a significantly more useful level of functionality than sending simple motor commands since the motion is performed locally and autonomously by the navigation stack — the full functionality of ROS comes into play, for instance obstacles are detected and the robot will stop until the obstacle is removed.

In practice we found that unless the remote user is familiar with the environment where the robot is operating it is quite easy to become disoriented or lost. A map is a very useful tool for the remote user and of course the robot navigation stack has a map. The handler for the "getmap" chat command obtains a map from the ROS navigation stack and then uses Skype's file transfer capability to push that map to the remote user as a GIF image. By default the ROS maps are stored in a 4000×4000 pixel image (5cm grid cell size) which is cropped before sending — typically less than 20% of this area contains useful information. Figure 2 shows the remote user's desktop with the Skype client showing the navigation camera image, the text chat window, and a map sent by

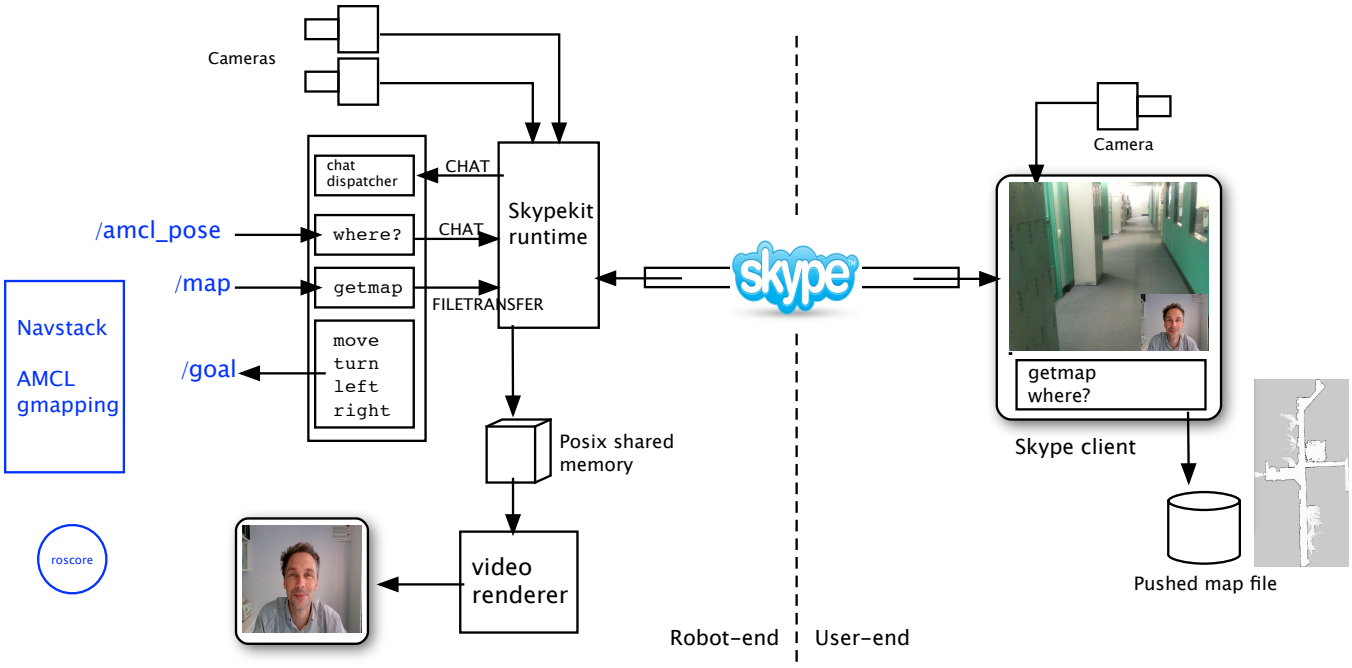

Fig. 3. System architecture for chat-based navigation. ROS related functionality is shown in blue.

the "getmap" command. The "where?" command reports the robot's current pose, with respect to the map origin, in numeric form.

The Skypkit runtime logs into the Skype network as a unique user, picks up the incoming call, passes local audio and video to the remote user, and renders incoming audio. As mentioned earlier Skypekit has no GUI and cannot display the video feed from the remote user. For Unix-like systems the video frames are written into Posix shared memory for access by a rendering client. We wrote a separate C++/Qt program to perform video rendering from the shared memory buffer.

III. Map-based navigation

While the map was a very significant aid for navigation it was still clunky to issue "where?" commands and mentally plot the location on the map. We wanted to create an interface akin to the GPS navigation system in a car — to display a map and our position on that map — since we know this is a very useful tool when navigating in an unknown city. This required providing the user with a richer connection to the robot's navigation stack — using the robot's pose to animate an icon on a map on the user's desktop, and to allow the user to drive by clicking points on the map.

To implement this meant extending the robot-end server and using Skypekit at the client end as well, as shown in Figure 4. Rather than chat, since the communications is program to program, we use Skype's App2App messaging for the following tasks:

- to request a map from the robot-end server,
- to push robot pose updates to the user-end map display, and

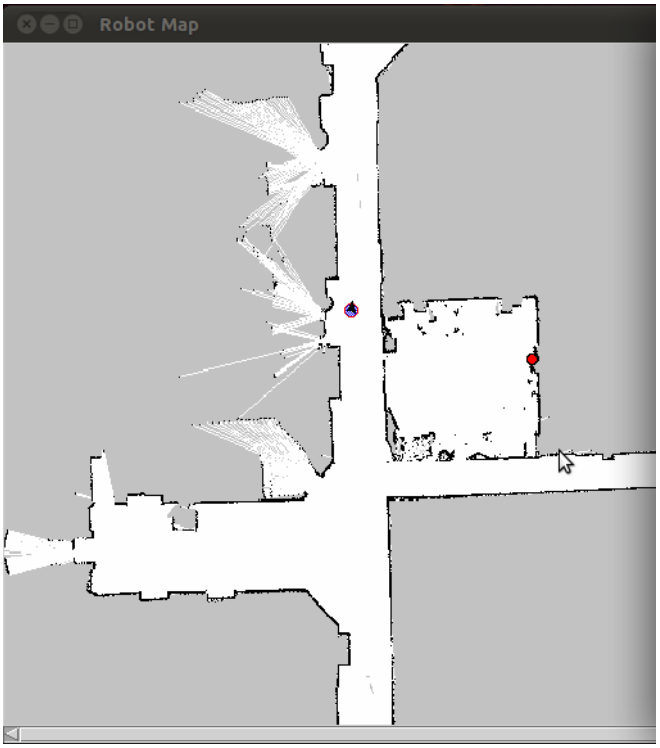

Fig. 5. Remote map view for map-based navigation. The robot's current pose is shown as a circle with a direction indicator (shown in the middle of the long vertical corridor). The red dot to the right is the origin of the robot's map.

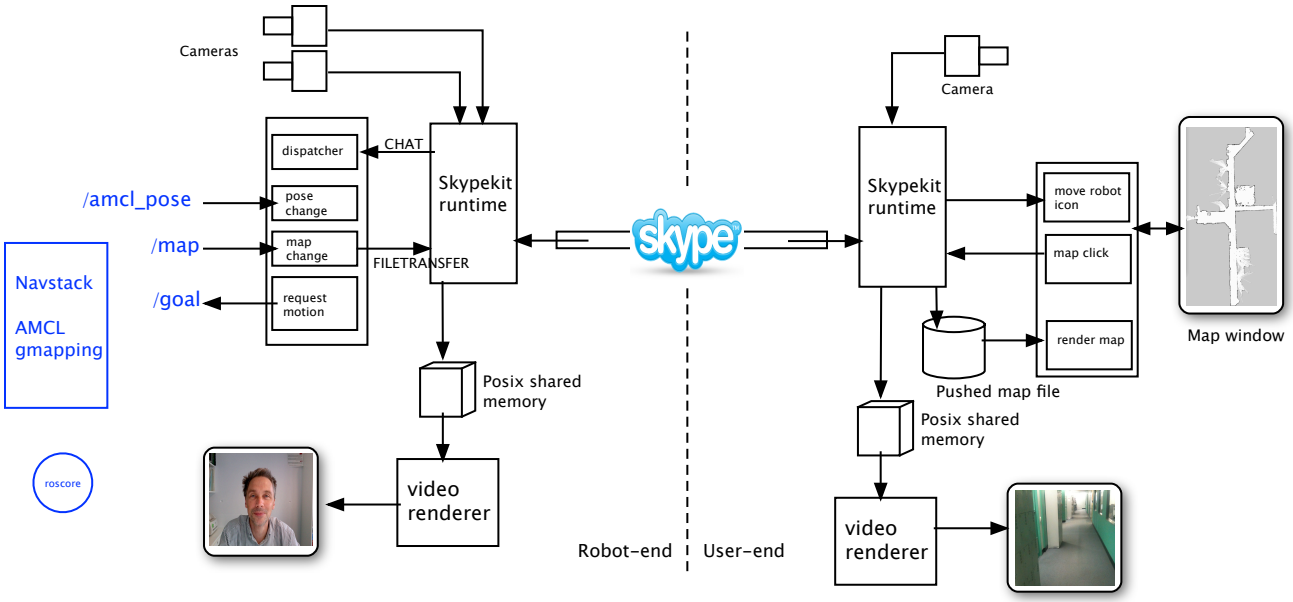

Fig. 4. System architecture for map-based navigation. ROS related functionality is shown in blue.

- to push navigation goals to the robot-end motion server.

This structure is easily extensible to other information such as battery levels, robot moving status, ROS diagnostics and other utility information which could also be displayed on the user-end GUI.

To ensure maximum portability our intention was to write the user-end application in Java but the Java API for SkypeKit is currently rudimentary. Instead we used the Python API and the TkInter graphical interface which is a standard part of all Python distributions, and in our experience less problematic across platforms than WxPython. The program is event driven and handles events from ROS, Skypekit and the TkInter windowing system.

When the user-end client starts it requests a map which is delivered by Skype file transfer. An event handler on file transfer completion causes the file to be rendered into a window. ROS pose updates on the /amcl_pose topic are converted to App2App messages and sent to the user-end where an event handler updates the position and orientation of the robot icon on the map, see Figure 5. When the user clicks a point on the map an event handler sends an App2App message to the robot-end where a handler generates a ROS /goal topic.

A video renderer process is required to display the video from the robot-end.

IV. Safety and Security

For remote robot operation safety and security are critical considerations. Taking security first, the challenge is to prevent unauthorized users from anywhere on the internet taking control of the robot. For this we rely on Skype's security features of only accepting calls from users on the contact list. This in turn relies on Skype's robust authentication and requires the use of strong passwords for users who are on the robot's contact list.

To ensure safety we have a number of strategies. Firstly we rely on the robot's navigation ability to move in areas of free space, and the robot's onboard sensors (laser, ultrasonics and bump sensors) to stop in the presence of obstacles. We limit the robot's speed and ensure that the vision stream from the robot-end shows the area in front of the robot while the robot is moving — we want to discourage a remote-user from walking and talking.

If the user hits any key in the map GUI a message will be sent to stop the robot immediately. If the robot-end Skypekit detects that the call has been hungup or become disconnected the appropriate callback functions will command the robot to stop.

V. Reflections

A. Portability

We made the decision to use Skypekit rather than the vanilla Skype client since Skypekit is actively supported by Skype and has an API with Python bindings [13]. Python programs are quite portable and provides an easy integration path with ROS. However in retrospect the Skypekit path was more difficult than we had expected: to gain access to Skypekit requires joining the developer program; the runtime is only available for desktop platforms and must be specifically requested from Skype; a keypair is required to allow it to operate; and a separate application is required to display video though C++/Qt is quite portable. The result is three applications (Skypekit runtime, Python application and C++/Qt video renderer) at the user-end which is more complex than ideal.

The alternative is to use a vanilla Skype client at the user-end and remote control it through the Desktop API [12]

which is quite complete but there are no Python bindings and the interface is operating system specific. Under Linux the interface is via DBUS which can be interfaced to Python using the dbus module. Under MacOS the interface is a Cocoa framework which requires an application written in Objective C — applications can be written in Python using PyObjC and then translated into a MacOS application. For both Linux and MacOS this requires non standard Python modules with unknown support status.

B. Future work

The framework we have developed is powerful and opens up many opportunities and we discuss some of them below:

a) Voice feedback: for the chat-based interface we are testing the integration of voice synthesis so that the robot-end server responds by voice rather than by chat.

b) Robot teleconferencing: where one user can interact with and through a number of robots.

c) Robot-end person recognition: where a Kinect sensor and/or face recognition alerts the remote user of the presence of a person. Kinect-based person and gesture recognition will also allow for a "follow me" primitive where the robot follows a local person to some destination rather having to be remotely driven. This would allow for a safe remote walk and talk behaviour.

d) ROS topic transport: Skype's App2App messaging allows an arbitrary number of datagrams to be sent to and from named endpoints and this has a very strong similarity to ROS topics. It is possible for the user-end client to request the robot-end to subscribe to an arbitrary ROS topic and send topic updates on a new App2App stream to the user-end where they could be rendered.

This could be taken further with the development of a user-end ROS proxy that would relay ROS topics from the robot-end via multiple App2App message streams. This would allow the user to run more sophisticated ROS applications such as rviz and use Skype as the data transport layer, with all the advantages this has for both security and firewall penetration.

e) MATLAB integration: In Section II it was mentioned that Skypkit does not perform video display, but instead writes video frames into Posix shared memory. This has the advantage that the incoming video stream can be shared by multiple clients in addition to a video display window. We have written a simple video logger and also a MATLAB mex-file that brings frames into the workspace. This allows image-based control of the remote robot using floor color segmentation or vanishing lines. Working in MATLAB allowed us to use the Machine Vision Toolbox [14] as well as MATLAB's extensive suite of data visualization tools.

VI. Conclusions

Skype is a well known tool for tele- and video-conferencing but it has little-known but powerful capabilities for integration with other software which make it well suited for use as a powerful framework for robotic teleoperation and telepresence. Some of its advantages include connectivity through firewalls and NAT boxes, powerful communications primitives for multiplexing application data streams alongside audio, video and chat data, and strong security. We have integrated Skype into a ROS/Linux framework which gives the remote user all the advantages of local robot autonomy such as map-based navigation and obstacle avoidance. This allows the remote user to not only interact with people near the robot but also to view maps, robot sensory data, robot pose and to issue high-level motion commands to the robot's navigation stack.

References

[1] F. Michaud, P. Boissy, H. Corriveau, A. Grant, M. Lauria, D. Labonte, R. Cloutier, M. Roux, M. Royer, and D. Iannuzzi, "Telepresence robot for home care assistance," in *AAAI Spring Symposium on Multidisciplinary Collaboration for Socially Assistive Robotics*, 2007.

[2] H. Nakanishi, Y. Murakami, D. Nogami, and H. Ishiguro, "Minimum movement matters: impact of robot-mounted cameras on social telepresence," in *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, ser. CSCW '08. New York, NY, USA: ACM, 2008, pp. 303–312. [Online]. Available: http://doi.acm.org/10.1145/1460563.1460614

[3] Robot operating system. [Online]. Available: http://www.ros.org

[4] J. Drury, B. Keyes, and H. Yanco, "Lassoing hri: analyzing situation awareness in map-centric and video-centric interfaces," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*. ACM, 2007, pp. 279–286.

[5] iRobot Corporation. irobot ava mobile robotics platform. [Online]. Available: http://www.irobot.com/ava/

[6] (2011) Anybots. [Online]. Available: https://www.anybots.com

[7] J. C. Lee. (2011) Low cost video chat robot v2. [Online]. Available: http://procrastineering.blogspot.com.au/2011/02/low-cost-video-chat-robot.html

[8] ——. (2011) Low cost video chat robot. [Online]. Available: http://youtu.be/9LNS9CivO34

[9] (2012) Sparky jr project. [Online]. Available: http://sparkyjr.ning.com/

[10] D. Schneider, "I, office worker [hands on]," *Spectrum, IEEE*, vol. 47, no. 10, pp. 20 –21, october 2010.

[11] H. Wiguna. Skyduino. [Online]. Available: http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1266383955

[12] Skype public api. [Online]. Available: http://developer.skype.com/public-api-reference

[13] Skypekit api. [Online]. Available: http://developer.skype.com/skypekit/reference/

[14] P. Corke. (2012) Robotics, vision & control toolboxes. [Online]. Available: http://www.petercorke.com/RVC/top/toolboxes/