Building a Semi-Autonomous Sociable Robot Platform for Robust Interpersonal Telecommunication

by

Robert Lopez Toscano

S.B. Computer Science and Engineering, M.I.T., 2007

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

May, 2008

Author_____

Department of Electrical Engineering and Computer Science

May 26, 2008

Certified by_____

Dr. Cynthia Breazeal

Associate Professor of MAS, MIT Media Lab

Thesis Supervisor

Accepted by_____

Arthur C. Smith

Professor of Electrical Engineering

Chairman, Department Committee on Graduate Theses

Building a Semi-Autonomous Sociable Robot Platform for Robust Interpersonal Telecommunication

by

Robert Lopez Toscano

Submitted to the Department of Electrical Engineering and Computer Science

May 25, 2008

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis presents the design of a software platform for the Huggable project. The Huggable is a new kind of robotic companion being developed at the MIT Media Lab for health care, education, entertainment and social communication applications. This work focuses on the social communication application as it pertains to using a semi-autonomous robotic avatar in a remote environment. The software platform consists of an extensible and robust distributed software system that connects a remote human puppeteer to the Huggable robot via internet. The paper discusses design decisions made in building the software platform and describes the technologies created for the social communication application. An informal trial of the system reveals how the system's puppeteering interface can be improved, and pinpoints where performance enhancements are needed for this particular application.

Thesis Supervisor: Dr. Cynthia Breazeal

Title: Associate Professor in MAS, MIT Media Lab

Table of Contents

Introduction

As the world's markets grow people need to travel farther and stay for longer away from their friends and families. Soldiers, away in foreign countries, are separated from their families for months at a time. Through our desire to stay in touch with our loved ones, telephones and cell phones were developed. However, just being able to hear the person on the other end of this communication channel was not enough. With the advent of the internet and its wide adoption, video conferencing software like NetMeeting, Yahoo Messenger, and Skype have been able to connect people through audio and video. Even still, these technologies lack a fundamental part of human communication--sharing a physical space. Humans can share a physical space in many ways. They can embrace each other, give each other objects, or point at things in the same space. So far, these purely software based communication applications have not achieved this. However, a series of physical communication technologies have been developed to fill this fundamental gap.

Physical Communication

The Hug Shirt is a wearable shirt with embedded sensors that can sense strength of touch, skin warmth, and heart rate and can send this data over a distance to another Hug Shirt wearer where built-in actuators recreate the sensation of touch, warmth, and emotion of the hug. It works by sending the sensor data via Bluetooth to one's cellular phone and then sends that information to another cellular phone to be recreated as a hug [7]. In this product we see an attempt to improve electronic communication between humans by sending not just knowledge but human gestures.

The Hug is a conceptual robotic product designed to facilitate intimate communication across distance, emphasizing the physical aspects of that communication. The Hug was developed at Carnegie Mellon University to experiment with the design of "robotic products". A major motivation of this project was the realization that the form of a "robotic product" has major effects on its capabilities and context of use. For example shaping this

product like a child could invoke feelings similar to that of hugging a real child [8].

The inTouch project is a device that is a medium for haptic interpersonal communication. The idea behind the project is to create a device that would allow two users to feel like they are manipulating the same object. The object chosen for their prototype was a set of rollers that could move when a user or her partner moved their respective rollers. There are two sets of rollers connected by cabling yet they move as one. Informal user testing illustrated that users indicated interest in the shared manipulation of the device and often described the interaction as playful [3]. This is another attempt at conveying the physical aspects of communication through technology.

Robot-Mediated Communication

The field of social robotics has introduced the robot as a social player. Robots are now being designed to interact in the same social space as humans do. In Cynthia Breazeal's book, *Designing Sociable Robots* [4], she writes about building Kismet, a robot that was designed to evoke an emotional and sympathetic response from people who interacted with it. In the industry of communication, robots can provide us with the idea of telepresence. Telepresence consists of technologies enabling a user to feel his or her presence in a different location from his or her true location. Using robots to mediate communication between two human parties, researchers can leverage both of these ideas--social engagement and telepresence--in order to create a rich communication experience. Work building on these ideas has been done with a variety of different types of robots.

Cory D. Kidd's (a Ph.D. student at the MIT Media Lab) thesis, "Sociable Robots: The Role of Presence and Task in Human-Robot Interaction", explores the idea of robots as social partners. His thesis analyzes robotic applications in entertainment, education, and healthcare where the robot can be perceived to be trusting, helpful, reliable, and engaging. His observations of human interactions with a robot, animated character, and another human show that interaction with a robot is qualitatively more alive, more real, more engaging, and produces more real emotions in a user than an animated character, but less

so than for a real human. He concludes that the physical existence of the robot causes feelings of engagement and social presence [15]. These findings suggest that a robotic interface for human-computer interaction is more effective at social communication than a virtual or animated character.

Hiroshi Ishiguro, a senior researcher at ATR Intelligent Robotics and Communication Laboratories outside Kyoto, Japan, has created an android (Geminoid HI-1) who looks and moves just as Ishiguro does. The android has the capability of being remotely controlled from his home where he can give his class lessons through the android while skipping the commute to work. Ishiguro's intent was to explore the idea of "tele-interaction" and give the robot "presence". People who have interacted with the android hesitate to even poke the machine's rubbery hands and cheeks [14]. While this thesis will not attempt to create a human-like robot, it will further investigate the idea of "telepresence" that has already been explored in Ishiguro's work.

Other robots designed to investigate the idea of telepresence include Robonaut [11], Sony's AIBO [23], Quasi [13], and Disney Imagineering's Muppet Mobile Labs [33]. Goza et al. had developed a teleoperation system for Robonaut consisting of VR helmet displays, body posture tracking Polhemus™ sensors, and a finger tracking Cyberglove™. Although, Goza's system provides a full puppeteering system for the robot, it is inappropriate to use in people's homes, unlike common video conferencing software.

These research projects suggest that there is substantial grounding for the pursuit of using robots or robotic devices to facilitate the physical aspects of communication as well as engage the user through social expression. While these are all sophisticated robot systems, their use in a communication scenario relies primarily upon only two senses – vision and audio. I believe that the social communication aspects of these systems can be greatly improved by allowing the puppeteer to see more than just vision and audio, but also understand how the robot is being physically touched, held, or interacted with. Additionally, by adding layers of autonomy on top of the traditionally teleoperated robot we can reduce the cognitive load of the puppeteer while improving the overall interaction experience for the user. This puppeteered robot then becomes a semi-autonomous robotic avatar that

serves to establish the puppeteer's presense in a remote space.

The Huggable

For the past three years, the Personal Robotics Group at the MIT Media Lab have been developing the Huggable robot platform described in [25]. The Huggable, shown in figure 1, is a new type of robotic companion designed to function both as a fully autonomous robot as well as a semi-autonomous robot avatar.

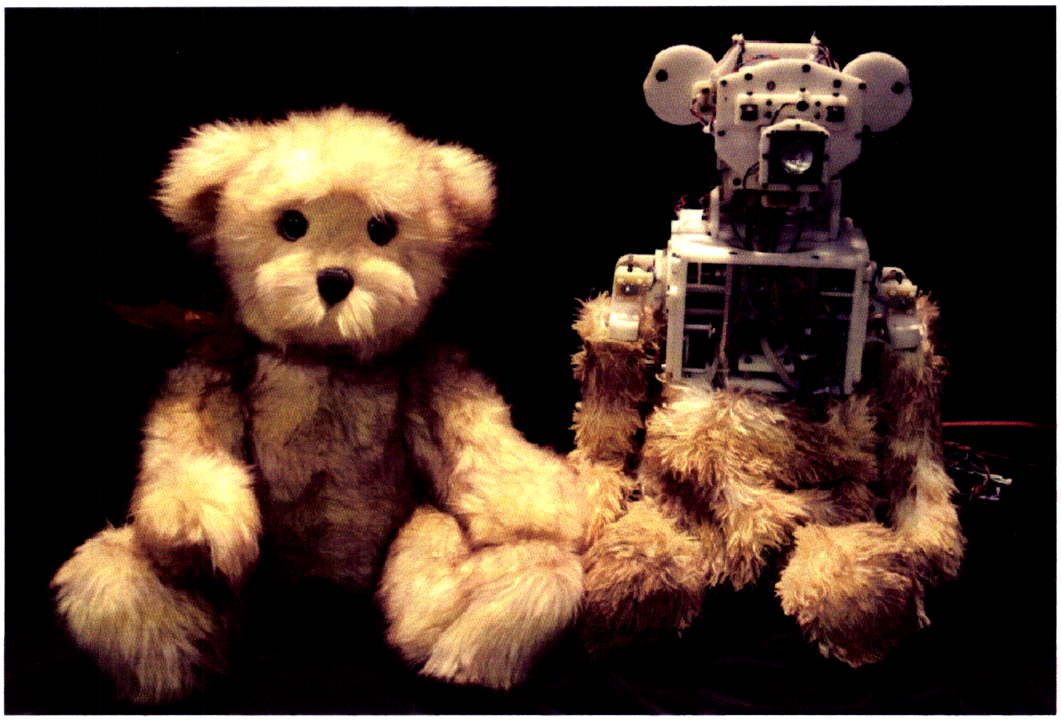


Figure 1: A photograph of the concept plush teddy-bear (left) and the robot in development (right). Notice the microphones in the robot's ears, pin-hole cameras for its eyes, and speaker in its snout.

Underneath its soft plush teddy bear exterior and silicone skin, the Huggable is being designed with a full-body, multi-modal "sensitive skin" [26], two cameras in its eyes--one color and one black and white, a microphone array in its head and ears, an inertial measurement unit (IMU) in its body [25], a speaker in its snout, potentiometers to detect joint angle positions, and an embedded PC with wireless networking. The robot has a total

of 8 degrees of freedom (DOFs) : a 3 DOF neck (for nodding, tilting, and rotating), a 2 DOF shoulder motion (up/down and in/out) per arm, and a 1 DOF ear mechanism for expression. The Huggable also uses a hybrid belt-gear mechanical drive system which allows for smooth and quiet motion. Currently, the robot is tethered to a 12V power supply, but ultimately will run on battery power.

For the sake of clarity I'd like to define the participants in this application. The Huggable system consists of three entities--the robot, the puppeteer, and the human that is interacting face-to-face with the robot. This last person I will call the user. I make no assumptions about the level of skill that the puppeteer possesses and the user can be anyone from a child to an adult. The puppeteer is situated in a remote location given only a computer with microphone, web camera, and internet connection. The puppeteer is intended to be remote enough from the physical locality of the robot that the puppeteer can not see, hear nor otherwise naturally sense the robot. The puppeteer's only input from the robot is given through the user interface on his or her computer. The user is situated near the robot (usually in the same room) and can physically interact with it.

For this thesis, I designed and implemented a software platform that would multiplex and process the data from the multitude of hardware sensors on the robot as well as present this processed information to the puppeteer of the robot. While the platform was designed to be general enough to function in many applications other than the social communication one, some of the technologies designed and implemented on top of the Huggable software platform were specifically designed to tackle problems in the social communication application.

Problem Statement

In the following section, I outline two sets of requirements for the Huggable's software system. The first deals with requirements for the social communication application and the second concerns the computer engineering problems associated with building a robotic

software platform.

Social Communication Requirements

The social communication application necessitates six requirements of its underlying implementation. First, the robot must feature systems which allow for the remote puppeteer to direct the attention of the user or be capable of responding to the users' own attempts to direct the robot's attention. Second, both the puppeteer and user should be able to share attention easily, i.e. both user and robot can interact with and focus on the same object. These two features play an important role in our social communication application. When the user reads a book together with the robot, either the user or the robot may point at a specific figure or sentence in a book. To enable such features, the embodiment aspect of the robot combined with the puppeteer's ability to directly control its arms and head allow the user to recognize where the robot is gazing and/or pointing at.

Third, the robot must provide the puppeteer with real-time multi-modal sensory information for situational awareness. The data must be presented in a clear, easily understood fashion that allows the puppeteer to be immersed in the interaction. This real-time sensor information may include the physical orientation of the robot, where and how the user is touching the robot, and other descriptive information to improve the interactive experience. Fourth, the robot must be controlled in such a way that reduces the cognitive load of the puppeteer while allowing for rich forms of expression (vocalizations, facial expressions, gestures, etc.). Controlling a robot is still a cumbersome task, especially for elders. Many current control interfaces for robots remain difficult to learn and non-intuitive. For these reasons, making the interface as intuitive as possible by alleviating the cognitive load of the puppeteer is crucial.

Fifth, the robot's expressions and behavior must be readable to the user and convey personality to make the interaction fun, engaging, and personal. This may entail supporting the remote puppeteer's ability to convey his or her own personality through the robot avatar, or to control a robot to convey a consistent character (e.g., a robot that is based on

a familiar comic book character). This might include specific content such as sounds, gestures, and other behavioral elements typical of that character. Finally, the interface between puppeteer and the robot must be widely accessible, ideally from anywhere in the world. For instance, a World Wide Web interface would enable family members to interact with a child at great distances.

Software System Requirements

In addition to the six elements of design mandated by the specific application, there are a host of engineering problems associated with building a platform to support such design elements. Some of the difficult software engineering problems associated with building software platform for a robotic avatar that is meant to be puppeteered remotely over the internet are latency, reliability, and security. When connecting to the robotic avatar from a remote computer, the communication pathway through the internet might experience slowness due to a number of things ranging from hardware inadequacies to heavy user traffic. This problem of latency can significantly inhibit the real-time nature of puppeteering an expressive robot designed to interact with humans. Also, the reliability of the internet is inherently not guaranteed which leaves the developer with the need to accommodate lost data and/or out of order data that is communicated over the internet. Communicating over an internet protocol that does not address these issues might result in data losses in the visual and auditory feedback channels which might hinder the puppeteering of the robot. Finally, the openness of the internet creates a security risk of exposing the puppeteering interfaces that control the robot. The software platform for this type of application should ensure the privacy of the data transmitted between robot and puppeteer since some of this data can be personal media such as video and audio.

The final set of engineering problems I explore relates to building a robotic platform that will be adapted for novel applications in the future. This requirement mandates that the robotic software platform be extensible, scalable, and maintainable. The Huggable project's first goal is to produce a robotic platform that can be further developed to fit the need of a semi-autonomous social application. For this to be achieved, the software platform needs to be

extensible enough to easily build and integrate new technologies as well as adapt existing technologies such as computer vision and machine learning algorithms. Scalability must achieved to anticipate any kind of mass usage of the Huggable software platform or any kind of performance driven applications. Finally, to ensure the longevity of this software platform, the system must be easily maintained. Robotic software platforms have the unfortunate disadvantage that they heavily rely on hardware. Diagnostic tools and monitoring applications are crucial for easy maintenance and rapid development.

This thesis paper will present the research into these engineering problems: the design of the large-scale systems to be used by this social application, and the development of the technologies that solve the aforementioned problems in the robot-mediated communication domain.

Choosing a Robotic Software Platform

In general, a robotic software platform should be chosen to offer the following: a unified service execution environment, a set of reusable components, and a debugging and simulation environment [22]. In addition, this semi-autonomous social robotic avatar application calls for a few more--ability to perform well enough on the difficult platform of the internet, integrate well with existing technologies, and allow for the extensibility to other novel applications. Many robotics platforms such as Carnegie Mellon Navigation (CARMEN) Toolkit described in [26] and [27], Microsoft Robotics Studio (MSRS), and the software system developed for Stanley (the 2005 DARPA Grand Challenge winner) all emphasize distributed computing models. These distributed models usually consist of modeling the software system as a collection of independent *services* that can all run in the same process, different processes, or even on different computers across the network. In the Huggable software architecture, these *services* usually pertain to different sensors of the robot, or *services* that process incoming data from other *services*. One advantage of the distributed *service* approach is the reliability gained by isolating the different *services* from each other. Any adversities that any of the *services* encounter will not affect the other concurrent *services*. Another advantage of the distributed approach is the ability to offload

computation to other computers. While the previously mentioned software architectures all utilize a distributed approach to robotics software, some satisfy more of the other requirements than others.

A brief survey was conducted comparing the different software platforms: CARMEN, MSRS, and C5M (descibed in [5] and [2]). The feature sets of each system were reviewed keeping in mind the following criteria: performance, reliability, knowledge transfer, extensibility, and integration with existing technologies. Consequently, the MSRS software platform was chosen to be the basis of the Huggable project's software system according to the following reasons. Since most of the thesis work was intended for designing the system to fit the application of social robotic avatars and not trying to learn an entirely new framework, the primary criterion for choosing a platform was knowledge transfer--transfer of prior knowledge of a common development platform.

Managed languages such as Java, and Python are making their way into academia as the primary teaching languages for computer science. Their garbage collectors vastly reduce programming errors due to memory maintenance. Their virtual machines and just-in-time compiling have the potential to even surpass some static compile-time optimizations. For these reasons, a software platform built upon and implemented with a managed language and associated run-time was preferred. C5M and a particular implementation of CARMEN is written in Java, and MSRS is written in C#, both of which are managed languages. Applications built with C# run on Microsoft's .NET framework which is widely adopted for Windows-based development in the computer software industry. The framework and its accompanying libraries have been optimized for performance and security. The security features of the .NET framework are relevant especially to a robot that can be remotely puppeteered. Protection from common attacks such as stack smashing and buffer overflows can help secure access to controlling the robot.

One of the key requirements for a robotics software system to be extensible is that the independent *services* within the system be as loosley coupled as possible. In MSRS, each *service* is compiled into its respective binary file. Dependencies between *services* are established via a second binary file called a proxy. This proxy does not contain any of the

implementation code from its respective *service* but contains stubs of the *service*'s public application programming interface (API). This allows a *service* to let other dependent *services* make use of its API while maintaining the freedom to change the implementation of any of its public interfaces without having to recompile any of the dependent *services*. This feature is invaluable for parallel development, especially in robotics since there is usually a wide variety of technologies used in the robot that can sometimes span the collective expertise of a group of developers. Another advantage to *services* being loosely coupled from each other is that *services* become very *plug and play* like. *Services* themselves, not the overall architecture, become the building blocks of the robotics software system. It would be very difficult to adapt the C5M behavior system for new applications since *services* within it interact closely with each other and would have to be redesigned if needed for different applications. Another advantage the MSRS architecture has over the C5M and CARMEN architectures is that the protocol for passing data between each *service* is well defined. In C5M and CARMEN, transport of data is controlled by the developer--whether it be over the network or over shared memory. Because MSRS abstracts the communication between *services*, MSRS can automatically switch between implementations of message transport depending on whether the two communicating *services* are in the same process, different processes, or are on different computers and hence need to communicate over the network layer. This allows *services* to be used in a variety of new ways that it may have not originally been intended for.

While many of these robotics software platforms are sufficiently general to support most imaginable applications of robotics, one arena to begin comparing the platforms further is to review what *services* the platform has already implemented for the developer. Examples of these types of *services* could range from *services* that perform some common localization and mapping algorithms to *services* that interface with common robot sensors. Since one of the problem domains of this thesis is designing a robot-puppeteer interface, any kind of user interface tools that the platform could already provide would make it a much more attractive one. MSRS has such tools. Since the MSRS communication protocol is built upon the Hypertext Transport Protocol (HTTP), the architecture integrates nicely with web-based interfaces. In turn, wide-spread web-based user interface techniques, such as the asynchronous JavaScript and XML (AJAX) [1] methodology can be employed to create easily

accessible web-interfaces to control the robot from a web browser on one's own computer.

Hardware System Details

Robotic Hardware

The robotic platform for the Huggable project is an eighteen inch furry teddy bear robot. It has been in development for over three years. The robot has pin-hole cameras in its eyes, microphones embedded in its ears, a speaker in its mouth, potentiometers in all of its movable joints to sense joint position, an IMU, an on-board embedded computer running Windows Embedded with 802.11 wireless capability, and a system of quiet actuated and back drivable motors that control the eight DOFs in the robot (see figure 2). The robot's head has three degrees of freedom--it can move its head up or down, left or right, or tilt its head from side to side. Each arm has two degrees of freedom which allow them to move up and down and rotate about. And lastly, the robot can wiggle its ears up and down. A full-body sensate skin is currently in development by other members of the Huggable project but a demonstrative prototype has been built that consists of a bear-shaped foam doll with 70 basic electric field sensors all over its head, body, arms, and legs.

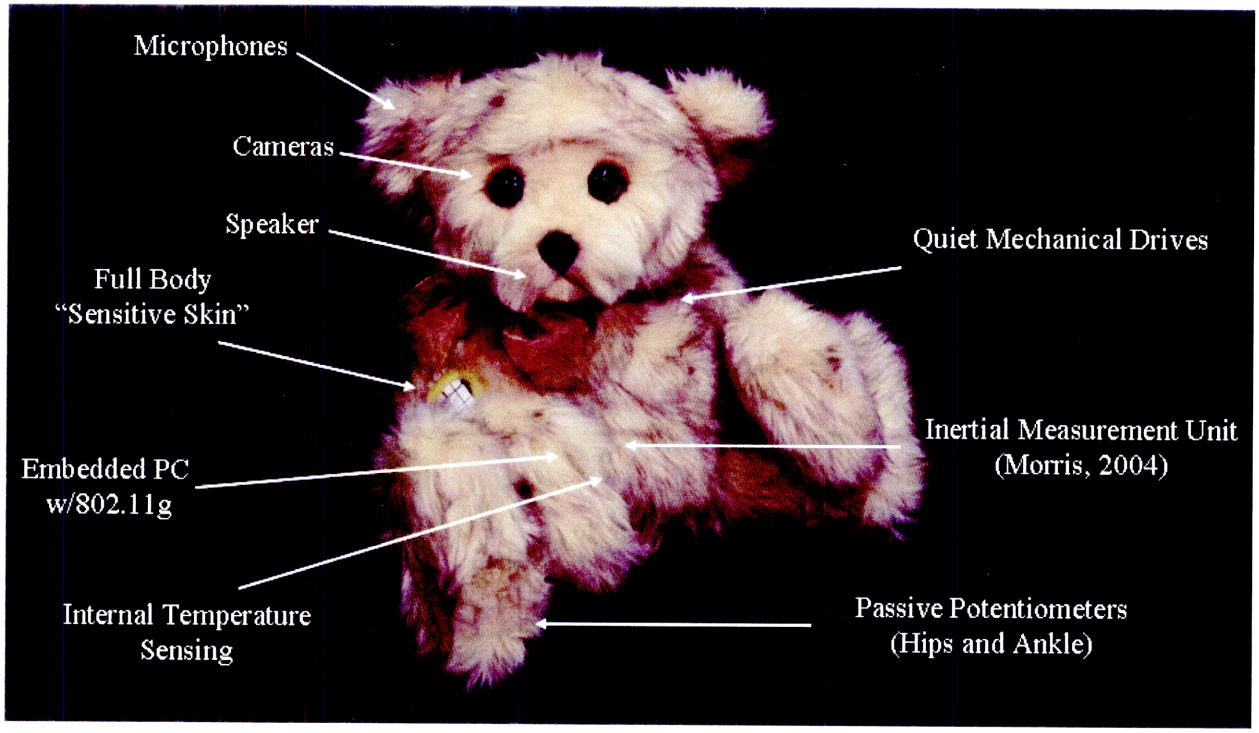


Figure 2: This figure identifies each of the hardware sensors present in the robot and approximately where they are located.

Computer Hardware

The Huggable platform uses a set of computers for the development and testing of the Huggable platform. The embedded computer inside the robot contains a 1.8Ghz Intel Pentium M processor with 1GB of memory and a 32GB Samsung solid-state drive. It provides 2 serial and 4 USB ports to gather data from the various sensor's on the robot. The same model Sager laptop is used to represent the user's and puppeteer's computers. The Sager laptops contain Intel Core 2 Quad processors at 2.66GHz each and 2GB of DDR2 SDRAM. An Apple laptop is used to run the C5M behavior system. It is an Apple MacBook Pro with an Intel Pentium Core 2 Duo processor at 2.33GHz and 2GB of memory. The Apple MacBook Pro runs the OSX operating system and the rest of the computers run Windows XP Service Pack 2.

The embedded computer and the user's computer (in the current implementation the user's computer is split into the MacBook Pro and one of the Sager laptops) are on the same network *subnet*. The remote puppeteer's computer (a Sager laptop) can be located in the same *subnet* or across the internet. For most of my development, the puppeteer's computer was in the same *subnet*.

Software System Details

The Huggable features a pair of software sub-systems to achieve its complex behavior. It uses MSRS version 1.5 running on the embedded computer to handle the gathering of data from the various sensors on the robot. This data is then forwarded to the user's computer (a Sager laptop also running MSRS) for heavy real-time processing. Some results of this processing are sent to the other software sub-system: C5M (running on the MacBook Pro). The rest of the results are sent to the puppeteer's computer (another Sager laptop), also running MSRS, to be displayed to the puppeteer. The C5M software sub-system uses the data it receives to make high-level decisions about the robot's behavior, such as where to look, or how to move. The custom TCP protocol that MSRS uses allows for communication with computers beyond the local subnet. This is necessary for communication across the Internet between a remote puppeteer and the local user.

The C5M behavior system is a toolkit for designing synthetic brains for virtual or robotic bodies in dynamic, uncertain, complex environments. It supports real-time interaction with people and other agents, multiple forms of learning within and across subsystems, and sophisticated motor control for bodies with complex morphologies. The last feature is heavily used in the Huggable platform. An animator makes life-like animations (e.g. waving, sleeping, ear flicking) using a 3D virtual model of the robot and C5M is used to playback those animations on the physical robot using a differential motor controller that runs in a separate process.

MSRS provides two components for the robotics developer. They are the Concurrency and

Coordination Runtime (CCR) and the Decentralized Software Services (DSS). The CCR provides a runtime for running highly concurrent *services* within the same process. DSS consists of an application model that is based on the REST standard. In this model, each concurrent *service* in the runtime is thought of as state machine that accepts messages from other *services* which will change its state, and it can send messages to other *services* to change their state. These *services* can run in the same process or be distributed across many computers.

There are many tools that MSRS provides to help developers build to the DSS application specification model. These tools are regularly used in the Huggable software platform. One tool that was mentioned before is the *DssProxy* tool which produces a proxy binary file containing only stubs of a *service's* acceptable messages. This *proxy* can be given to other *services* for compile-time checking of their code that sends messages to the *proxy* owner. Another useful tool MSRS provides is the ability to persist and load a *service's* state from an XML file. This was especially helpful when storing calibration values for each of the *services* that dealt with hardware sensors. Lastly, MSRS provides tools called *manifests*. These XML files define the startup state of an application developed for MSRS. In the *manifest* file, a developer can specify which *services* to start up, what their dependencies are (other *services*), and what initial state they will startup with. These tools have shaped the design and implementation of the rest of the Huggable platform.

A subset of the software technologies developed for the Huggable platform form the basis of this thesis. The first part of the engineering work for this thesis focuses on the technologies created to aid in the development and maintenance of the robotic software platform. More specifically, these technologies are designed to allow the developer to quickly diagnose problems, tune and calibrate the robot during run-time or at least without having to recompile. The latter part of this thesis focuses on the software developed that collects and processes the data coming from each of the robot's sensors as well as combining the processed data to produce a rich multi-modal user experience for both the puppeteer and the user. It will be shown that the latter thesis part fulfills the requirements for the social communication application outlined in the problem statement.

Designing a Robust and Extensible Framework

It was an important requirement for the software system of the Huggable project to be very robust and telling of any errors that it would encounter. MSRS offers a simple *service* for centralizing logging of messages within the same Distributed Software Services (DSS) node. A DSS node is process which hosts the MSRS runtime. Within this node, many *services* can run concurrently and independently. Communication between services is standardized--the same interface is used whether the communicating services are on the same node or on different ones. A base class offers the ability for any *service* to send messages to the logging *service* for storage. All logging messages from all *services* are stored in a xml-format file and can be viewed by a web browser. Another useful feature MSRS offers is that it runs each *service* in a sand-boxed type environment. If any one *service* encounters an error, or does not handle an exception, failure is isolated to the culprit service. All other *services* on that node are unaffected.

Use of MSRS in the Huggable Platform

As was mentioned before, the bulk of this thesis work was done on the MSRS platform. The atomic unit within the MSRS platform is the *service*. There are four types of *services* in the Huggable project. Some *services* are built for collecting data from the various sensors throughout the robot. These *services* I call the *producer services*, since they produce data for the rest of the system to process. The *producer services* gather data and broadcast it via MSRS's subscribe and publish API. *Processor services* then collect these data and perform computation ranging from filtering to classification. The collected data can either come from one *producer service* or can be multiplexed across multiple *producer services*. The third type of *service*, the *consumer service*, collects processed data from other services. A *consumer service* might collect data that belong to specific group, such as video related, or audio related. The purpose of the *consumer services* is really to interface with other parts of the Huggable software platform outside of MSRS. For example, some data collected by *consumer services* are shipped off to the C5M behavior system via the Inter-Robot Communications Protocol (IRCP) [12]. Other *consumer services* carefully prepare data to be

displayed to remote puppeteer. There are also cases where one particular *service* acts as two different types of *services* (i.e. a *producer* and a *processor*). The fourth and final *service* type is the *diagnostic service*. These services provide interfaces for technicians or developers to diagnose problems within the software at runtime, or calibrate and tweak sensors and other hardware. Again, these types of *services* can overlap with other *service* types.

Cross-computer communication is done through MSRS's custom communications protocol, Decentralized Software Services Protocol (DSSP). This allows *services* running on one computer to communicate with other *services* on a different computer in the same way that these *services* would communicate with *services* in the same process. On a side note, this was a very useful feature of MSRS that allowed us to experiment with different arrangements of these *services* to distribute the load as optimally as possible across the computers involved in the Huggable system. Other parts of the Huggable software system communicate through IRCP. An IRCP *service* was implemented to allow communication to flow from systems using IRCP to *services* running on MSRS. Figure 3 illustrates the entire system with all of its services.

System Layout

In this section, I outline all of the services that were developed for the Huggable platform by describing their function and their relationship to the rest of the *services*. I divide the services into three groups. Those that run on the puppeteer's computer, those that run on the user's computer, and those that run on the embedded computer. This grouping is very similar to the types of *services* I outlined earlier--*producer*, *processor*, and *consumer services*. Each service's relevance to the social communication application will be explained in later sections.

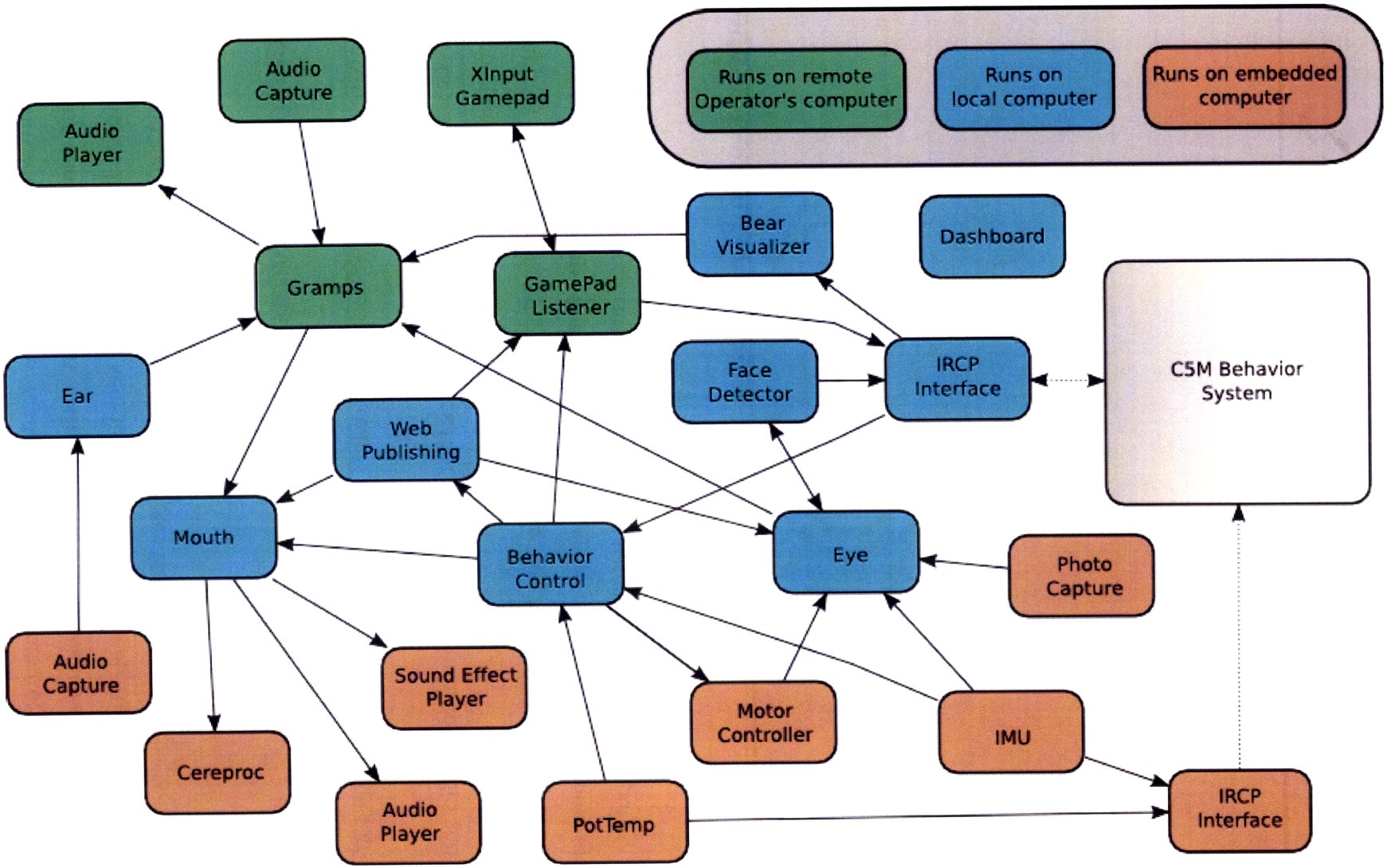

Runs on remote Operator's computer
Runs on local computer
Runs on embedded computer
Audio Capture
XInput Gamepad
Audio Player
Bear Visualizer
Dashboard
Gramps
GamePad Listener
C5M Behavior System
Face Detector
IRCP Interface
Ear
Web Publishing
Mouth
Behavior Control
Eye
Photo Capture
Audio Capture
Sound Effect Player
Motor Controller
IMU
Cereproc
Audio Player
PotTemp
IRCP Interface

Figure 3: This illustration shows which computers run what *services* and which *services* communicate with which other *services*. A solid arrow which points from *service* A to *service* B, signifies that A sends messages to B. Dashed arrows are messages sent outside of the MSRS DSS protocol. Instead, they are sent using the IRCP protocol. The floating *Dashboard service* sends messages to all other services but arrows were not drawn for clarity.

Multi-Computer Services

AudioCapture - a *producer service* that streams raw audio data from the computer's microphone.

AudioPlayer - a *consumer* low-level *service* that receives a stream of raw audio and plays it on the computer's speaker.

IRCPInterface - a *consumer* and *producer service* that can be used to receive or send IRCP packets including integers, floats, and byte arrays. For performance reasons, I have a separate instance of this *service* running on the embedded and the user's computer.

Embedded Computer Services

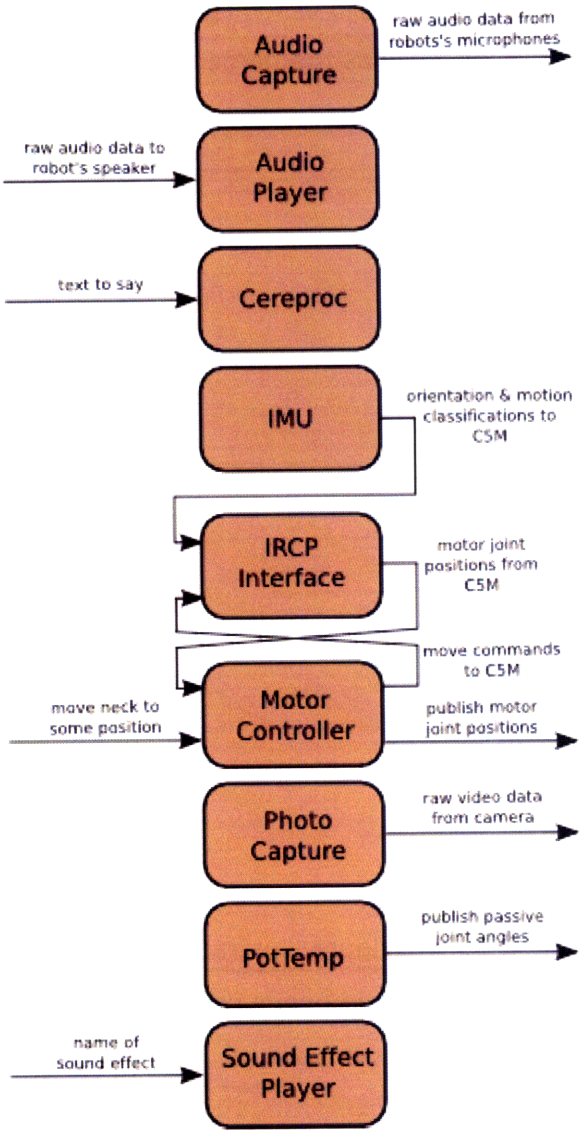


Figure 4: This figure illustrates the *services* that run on the embedded computer. Incoming arrows indicate what sorts of data the *service* receives, and outgoing arrows indicate what sorts of data are sent from the *service*.

Cereproc - a *consumer* low-level *service* that provides text-to-speech functionality via the Cereproc SDK [6].

IMU - a *producer service* that provides data from the inertial measurement unit sensor in the form of robot motion classification (an enumeration including, "bouncing", "rocking", etc.) and tilt orientation (in degrees).

PhotoCapture - a *producer service* that grabs, and makes available, frames from the video camera in the robot's eye.

PotTemp - a *producer service* that provides the positions of the potentiometers of the robot and provides a web interface for calibrating them. The "Temp" in the name refers to future

functionality which will read temperature data from different parts of the robot.

SoundEffectPlayer - a *consumer* low-level *service* that receives the name of a sound effect and plays the corresponding prerecorded .wav file.

User Computer Services

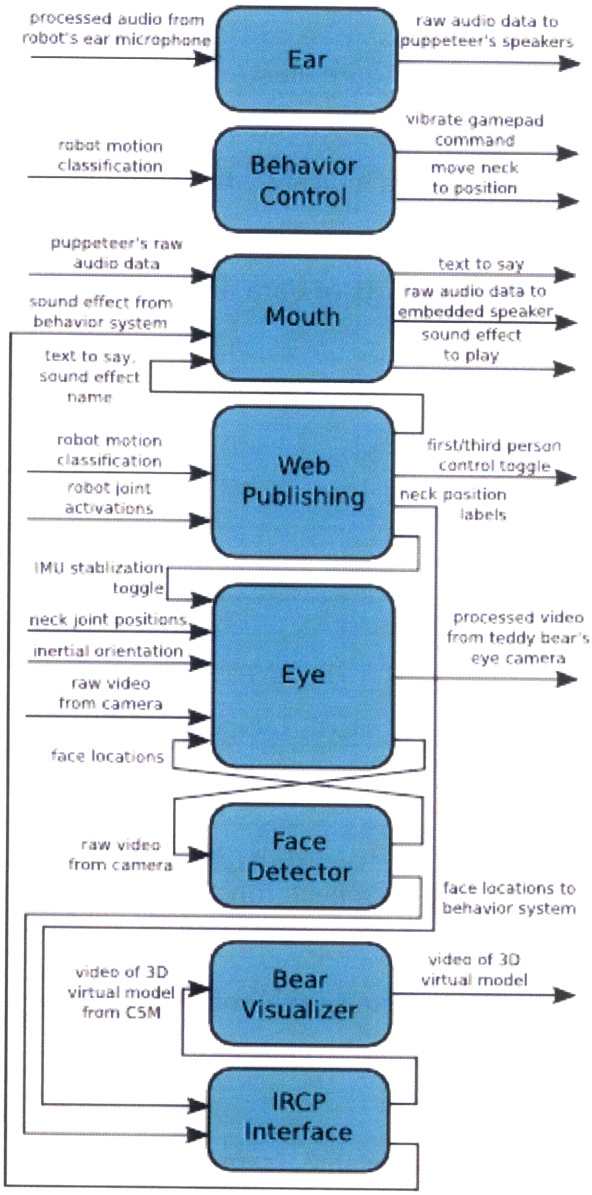


Figure 5: This figure illustrates which *services* run on the user's computer.

BearVisualizer - a *producer service* responsible for receiving, and making available, the frames of the virtual huggable video feed streaming from the C5M behavior system.

BehaviorControl - a *processor service* responsible for mediating input from the remote puppeteer destined for the robot's lower level *services* and making some of the state of the robot available for reading.

Ear - a *processor service* that mediates input from the *AudioCapture service* on the embedded computer and the rest of the system.

Eye - a *processor service* responsible for taking input from the *PhotoCapture* (raw video), *FaceDetection* (face locations), *IMU* (robot tilt orientation), MotorController (neck position),and *WebPublishing* (switch between upright vs. relative view) *services* and combining them to form a data-rich video feed for the remote puppeteer.

FaceDetector - a *processor service* that performs a face detection algorithm implemented by Intel's OpenCV library [19] on video frames, and returns the resulting locations and sizes of each face are returned to the requesting service.

Mouth - a *processor service* that multiplexes access to the robot's audio speaker including requests for the playing of raw audio, text-to-speech, and sound effects.

WebPublishing - a *consumer* and interface *service* which is a small web server serving the status of the huggable and accepting input from the user in the form of AJAX style get requests. From this website, a remote puppeteer can cause the robot to play sound effects, send it text to speak, view how the robot is being moved, change the camera upright/ relative view, flip the axis of the XBox 360 controller's joystick, choose what level of puppeteering they would like to perform (semi-autonomous/fully-autonomous), and label positions of the robot's neck for later recall.

Puppeteer Computer Services

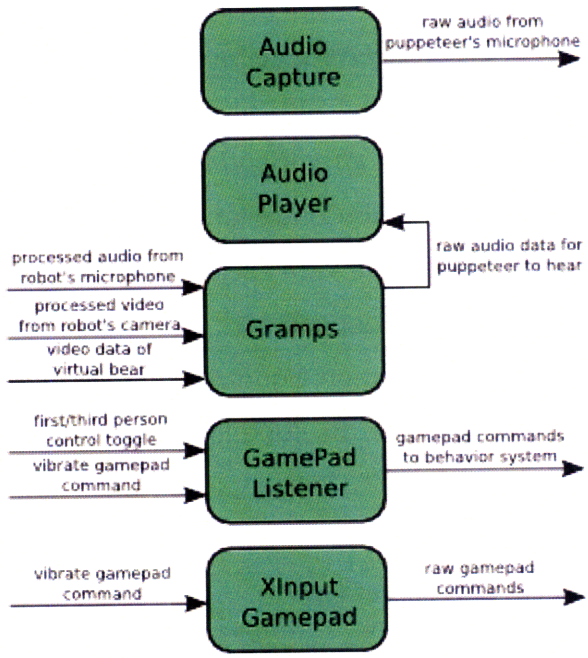


Figure 6: This figure illustrates which *services* run on the puppeteer's computer.

GamePadListener - a *processor service* that takes input from the XInputGamepad service and forwards it to the C5M behavior system (via IRCPInterface running on the user's computer) to control the robot's motors, aswell as transforming this data based on the point of view of control specified by the website interface (see WebPubilishing), and receiving vibration requests for the game controller.

Gramps - a *consumer* and interface *service* that displays video feeds from the robot's eye camera, and from the virtual robot model from the C5M behavior system. "Gramps" is an affectionate alias for the puppeteer.

XInputGamepad - a *producer service* bundled with MSRS that provides an interface to the XBox 360 controller.

Design Considerations

During the development of the Huggable software platform. I came across several frequently posed design questions that had to be answered with respect to the social communication application. The first is, which *services* should run on which computers?

What made the most logical sense was to place all data gather *services* (e.g. *IMU*, *PotTemp*, *PhotoCapture*, etc.) on the embedded computer since it would have all of the hardware devices plugged into it. It then made sense to put processing *services* (e.g. *Eye*, *Mouth*, *FaceDetector*, etc.) on the user's computer. This way, the embedded computer, which was the computational bottle-neck in our system, would have all of the computationally intensive processing offloaded to the user's computer. In addition to increasing the overall performance of the system, it also reduced the load on the embedded computer which in turn lowered power consumption, which could prolong battery life. The rest of the *services*, those concerned with presenting data to the puppeteer, ran on the puppeteer's computer. In practice, however, the embedded computer was overwhelmed with reading from the serial and USB ports. In order for the system to work at usable performance levels (i.e. the puppeteer can share and direct the user's attention, they can receive data fast enough to understand how the robot is being interacted with, etc.) I built throttling mechanisms in each of the *producer services*. This prevented streaming data like IMU orientation and potentiometer readings from overwhelming the CPU and allowing other services to run such as the playback of sound effects and text-to-speech. Also, some of the *producer services* were changed from a publish model to a only-on-request model, which brings me to the next design question.

The second design question that was frequently encountered was, under what circumstances should *services* adopt a push (or publish) rather than a pull (only-on-request) model for sending their data to other services. One advantage to the pull model would be that the *producer* would only have to do work when it was requested to by another *service*. This would help performance on the computer that was hosting the *producer*. The disadvantage would be that the *producer* needs to do some computation to retrieve the latest data from a sensor thus injecting more latency into the system. On the other hand, an advantage to the push model is that the *producer* does not get overwhelmed with requests from various *services* for its data. In the push model, when ever the data is ready, it is broadcast to all listening *services*. A disadvantage to the push model is that the *service* is always working at full capacity--at whatever rate data becomes available, hence increasing the computational load it contributes to the host computer. In the end, this design decision was made on a case by case basis given the nature of the data coming from

the *producer*, and given the restrictions of the environment that the *producer* was in. For example, I adopted the pull model for the *PhotoCapture* service for several reasons. Grabbing a frame from the camera was an expensive operation that taxed the CPU of the embedded computer. Also, and probably more influential, was the fact that the puppeteer could tolerate loss of video frames. This is not true for data such as audio, in which data gaps are harder to tolerate [23].

The C5M and MSRS platforms overlap in their functionality and scope. Another design question that was frequently encountered was, under what circumstances should a feature be implemented in C5M over MSRS and vice-versa. Like the other design questions, several issues had to be considered. Since the puppeteering interface could be located on a computer across the internet from the robot's network, any technology that needed to reach the puppeteer would have to communicate on a protocol that was reliable over the internet. This meant that DSSP (MSRS) was chosen over IRCP (C5M) since IRCP is designed for communication between modules on the same network *subnet.* Another consideration was latency. Since any data that needed to reach the puppeteer needed to go through DSSP, the easiest way to get to data from a feature in C5M to the puppeteer was to communicate the data to MSRS via IRCP, and then to the puppeteer via DSSP. Furthermore, the sensor data are collected by MSRS *services*, which meant that these data would have to be sent to C5M via IRCP for processing. These extra indirections introduced another source of latency between the robot and puppeteer. Given the above arguments, almost all new technologies that were built for the social application were developed on the MSRS platform.

The final frequently encountered design decision was never addressed fully in the work of this thesis. The robot possesses a significant degree of autonomy via the C5M behavior system but it was unclear when the best time to utilize this autonomy was (within the context of the social communication application). For example, a reflexive behavior was implemented that caused the robot to look at its feet when they were touched. During some early informal testing, it became bothersome that the robot would look away from the person of interest to its feet when the feet were unintentionally touched, or if something bumped into the table that the robot was on. In order to solve this specific example, some mechanism would have had to be developed that could detect when the puppeteer wanted

control and when he or she did not. Initial attempts at this solution involved providing the puppeteer with a radio button control on the website to allow the puppeteer to select the degree of control they wanted over the robot. For example, if the puppeteer chose, *full puppeteering*, the robot would no longer exhibit the aforementioned reflexive behavior. However, when the puppeteer wanted to turn the behavior back on, in order to be more responsive to the user, it was cumbersome to switch the radio button back during an interaction since the puppeteer's cognitive load was focused on the video feed or controlling the robot's DOFs. Proposed solutions to these autonomy problems are discussed further in the Improvements to the Huggable System section.

The HuggableServiceBase Class

While MSRS offers a helpful set of tools and *services*, it was necessary to additionally build application specific tools in order to improve the extensibility of the Huggable software system. The *HuggableServiceBase* base class is a subtype of the original *service* base class provided by MSRS. This base class offers several tools that perform tasks that are common throughout all *services* in this project. This base class provides a method that allows any *service* to register with the *Dashboardservice*. *Dashboard* will be discussed in more detail later in this section.

The base class also provides services for publishing the *service's* status to *Dashboard* by overriding existing logging methods provided by MSRS. This allows *services* that were not initially developed for the Huggable project to work seamlessly along-side existing Huggable *services*.

The base class also helps to abstract away the process of connecting to other *services* (or *partners*, as MSRS calls them). Out of the box, it is a lengthy coding process to dynamically connect one *service* to a *partner* in MSRS. One issue that had to be handled was that some *services* require some time to start up. There was no existing infrastructure in MSRS for a *service* to receive notifications of when a *partner service* came online, went offline, or halted due to error. Using this base class, a *service* can ask the *Dashboard* to notify the requesting

service if and when its *partner* came online and when the *partner* was ready. This makes it easy to deal with long complicated dependency chains between *services*. In addition, when a *service* has more information about the state of their *partners*, the *service* is empowered to make better decisions on how to procede with its own duties. For example, some *services* can work with or without data from its *partners*. With regard to connecting to other *services*, typical design of a Huggable *service* mandates that the dependent *service* try to connect to its *partner*, or *partners*, if possible, and if not, turn off its features that depend on data from those *partners*. This method of dynamically loading *services* allows developers to run *services* that just need to be tested in isolation or allows *services* to run even if it is impossible for a *partner* to also be running (i.e. the *partner* might need some unavailable physical hardware to run).

The *HuggableServiceBase* base class also handles messages that are sent from the *Dashboard* to all services in the Huggable software system, again enabling *services* developed for other applications to be integrated with the rest of the Huggable project with minimal code changes, if any. In short, the *HuggableServiceBase* class allows for the ease of creating new technologies for the Huggable project and integrates easily with existing infrastructure, while providing common functionality across all Huggable *services.*

The Dashboard Service

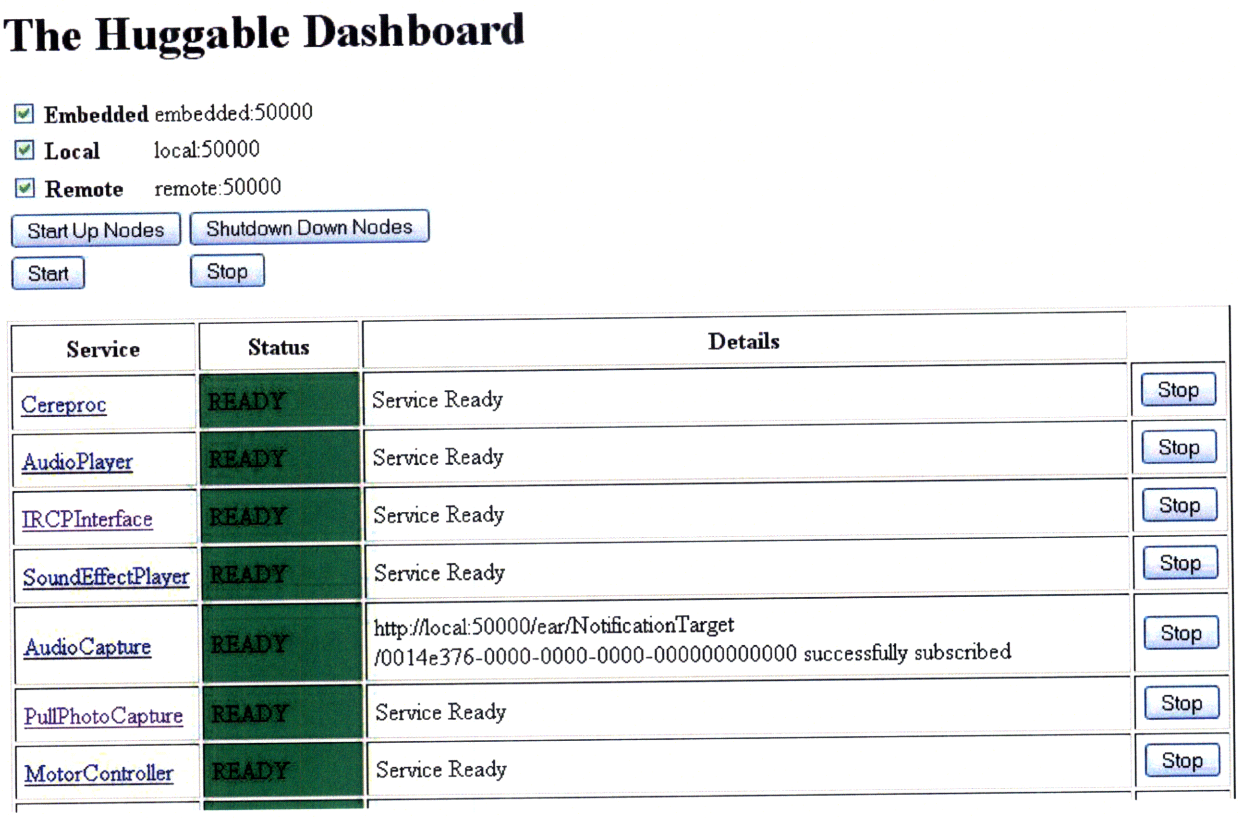


Figure 7: A screenshot of the *Dashboard* web interface. A technician can view the overall status of each of the *services* running in the Huggable software system from this one interface. The technician can also start and stop the entire system or just individual *services*.

The *Dashboard service* allows a Huggable project technician or developer to easily diagnose problems in the system quickly as well as provide an interface for the complicated process of starting and stopping the entire system (or parts of the system). Figure 7 shows what this interface looks like. The user interface is implemented using the AJAX method. MSRS offers tools for interacting with HTTP requests and serving XML as a response. *Dashboard* implements a handler for receiving HTTP *Get* messages (which *services* treat in the same way as any other message in MSRS), and returns the status of every *service* that has registered with the *Dashboard*. An XSLT file transforms the XML into HTML and subsequent

asynchronous HTTP *Get* requests made from a client-side script written in JavaScript, are sent to the *Dashboard* to receive updates to the HTML model, local to the browser.

Each *service* that registers with the *Dashboard* service is given a unique identifier (UID) that is composed of the *hostname* of the computer that the *service* is running on, the port that the DSS node is running on, and the name of the service. Figure 8 shows an example of one such UID.

```
http://embedded:50000/imu
```

Figure 8: An example of a UID of a *service* in the Huggable software system.

It was necessary to include more than just the name of the service as the UID since instances of the same service can run on different DSS nodes such as the *AudioCapture* and *AudioPlayer* services. It is probably worth noting here that this current design limits the number of DSS nodes that can run on a given computer to one, since the current UID does not distinguish between DSS nodes on the same computer. This UID format was chosen over other candidates because it was the same format used by the rest of MSRS and could potentially be used by the tools provided by MSRS. Using this UID, *services* can listen for status changes of their *partners* (identifying the *partners* by their respective UIDs) using a standard *Observer* design pattern [10]. A *service* obtains a UID to a *partner* by acquiring it from the manifest file that was used to start up the *service*.

Besides facilitating the initial connecting of *services* to one another, the *Dashboard* also monitors all *services* that have registered with it. Each *service* has a status and detailed message associated with it within *Dashboard*. The enumeration of possible states is:

- NOTREGISTERED - has not registered with the *Dashboard* yet
- STARTINGUP - has registered, but has not completed start up sequence
- READY - ready to accept messages
- OFFLINE - is no longer running and will not accept any messages
- OVERLOADED - has not responded to a *Ping* message for a specified timeout

- ERROR - has undergone a non-fatal error
- FATAL - has undergone a fatal error, the service can no longer accept any messages and needs to be restarted

After some time interval (specified in *Dashboard's* state file), *Dashboard* sends a *Ping* message to all registered *services*. The *HuggableServiceBase* class implements a handler for receiving *Ping* messages and simply posts a reply containing the *service's* UID back to *Dashboard*. If a *service* fails to reply to an outgoing *Ping* message after some time delay and for a certain number of *Ping* attempts (both are specified in *Dashboard's* state file), the Dashboard alerts the techinician via the web interface that it has lost connection to the failing *service*. The timeout and number of *Ping* attempts should be calibrated to account for the configuration of the computer setup. For example, the Huggable project's *Dashboard* makes 3 ping attempts and times out after 5 seconds. These numbers were arrived at by looking at the Huggable's computational bottlenecks. The embedded computer on the robot is the slowest of the system's computers, and it was empiracally measured that when the system was under normal usage, the *services* running on the embedded computer took up to five seconds to respond to a *Ping* message. If the *Dashboard* observes that any of the registered *services* fail to respond to a *Ping* message, the *Dashboard* marks that *service's* status as OVERLOADED. If the number of missed replies exceeds the maximum allowed, the *service's* status is changed to ERROR. This failure process is usually observed for a group of *services* if the computer that those *services* reside on shuts down or loses its connection to the network. Figure 9 illustrates what this would look like to a Huggable system developer. This feature of *Dashboard* was very useful in diagnosing these types of errors as well as quickly identifying where errors originated from. Several times, the *Dashboard* reminds the developer that certain hardware devices were forgotten to be plugged in or given power as is true in figure 10. Having this dynamic interface is much more convenient than reviewing a log file when the system is not working correctly. It also works better than watching print statements from console windows which is more difficult when on a multi-computer platform like the Huggable's.

AudioPlayer	READY	Service Ready
AudioCapture	READY	Service Ready
IRCPInterface	OVERLOADED	Service Ready
PotTemp	OVERLOADED	Service Ready
Imu	OVERLOADED	Service Ready

AudioPlayer	READY	Service Ready
AudioCapture	READY	Service Ready
IRCPInterface	ERROR	Service is unreachable
PotTemp	ERROR	Service is unreachable
Imu	ERROR	Service is unreachable

Figure 9: If a computer goes offline, the *services* that reside on that computer can no longer be *Ping*ed. The *Dashboard* will first assume that the *services* are busy and set their status to OVERLOADED, but after a few retries, *Dashboard* sets their status to ERROR.

AudioCapture	READY	http://local:50000/earNotificationTarget/00147067-0000-0000-0000-000000000000 successfully subscribed	Stop
PullPhotoCapture	FATAL	Failed to initialize capture device: System.Exception: No video capture devices found at that index! at MIT.Huggable.PullPhotoCapture.DirectShowLibBWCamera.InitializeCamera(Int32 deviceNum, Int32 frameRate, Size resolution) in C:\huggable-project\source\PullPhotoCapture\DirectShowLibBWCamera.cs:line 48 at MIT.Huggable.PullPhotoCapture.PullPhotoCaptureService.d__2.MoveNext() in C:\huggable-project\source\PullPhotoCapture\PullPhotoCapture.cs:line 98	Stop
MotorController	READY	Service Ready	Stop
BearVisualizer	READY	Service Ready	Stop
Mouth	READY	Service Ready	Stop
Ear	READY	Service Ready	Stop
FaceDetector	READY	Service Ready	Stop
Eye	FATAL	Could not connect to PhotoCapture partner: Partner, http://local:50000/pullphotocapture, in a fatal state.	Stop
WebPublishing	READY	Service Ready	Stop
BehaviorControl	READY	Service Ready	Stop
GamePadListener	READY	Service Ready	Stop

Figure 10: This instance of the *Dashboard* interface shows that the *PullPhotoCapture service* has halted because no camera was detected. Consequently, the *Eye service*, which depends on the *PullPhotoCapture service*, has halted as well.

Dashboard's last notable feature is its ability to start and stop the entire multi-computer system. A small custom program called *DssHostSpawner*, runs on all involved computers and listens on a fixed TCP port for network messages from the *Dashboard* service. When signaled, *DssHostSpawner* spawns a new process running a DSS node. That DSS node is then used to run all of the *services* that need to be run on that computer. An analogous stop signal causes the *DssHostSpawner* program to kill the DSS node process. The reason that a separate process is used to start and stop the DSS node process is that even when *services* are terminated by sending them a *Drop* message (which is standard procedure in MSRS), file locks on the *service's* binaries were not released, and so the binaries could not be replaced with a more updated version while the system was still running. Another reason was that during development, some *services* that use operating system resources would sometimes crash and corrupt the DSS node process, requiring that it be restarted. This *Dashboard* ability was crucial in speeding up the testing and development cycle.

The IRCPInterface Service

The *IRCPInterface* service plays a key role in making the Huggable software system extensible because it allows MSRS technologies to integrate with other technologies not built for MSRS but that support IRCP for their communication. My research group has developed several technologies for social applications of robots. All of these technologies were built using IRCP as the communication layer. They include the C5M behavior system and the Motor Controller. The Motor Controller is responsible for driving the motors of the robot and is implemented as a separate application that communicates over IRCP. A C# implementation of IRCP is wrapped in the *IRCPInterface service* and the *service* provides a MSRS-message-based interface for sending and receiving IRCP packets. To the MSRS *services* in the Huggable project, IRCP packets are seen as MSRS messages. The functionality that this *service* provides is invaluable since it frees the developer to use whatever platform he or she prefers and still be able to integrate with the Huggable software platform. However, it is encouraged to develop on the existing Huggable platform built on MSRS to utilize some of its tools that contribute to robustness and extensibility.

The C++/CLI Wrapping Method

Many vision-based and machine learning algorithms are written in the C or C++ programming languages such as Intel's OpenCV. Since these algorithms are used in the Huggable project a method was used to, as easily as possible, incorporate native C and C++ code from these algorithms and applications into the .NET runtime on which MSRS runs. The way this was accomplished was by using Microsoft's C++/CLI programming language. This programming language is visual C++ supplemented with .NET constructs in order for it to work with the memory managed and hightened security nature of a virtual machine like .NET. The killer feature of this language is the ability to mix native code and managed code in the same source file. This feature alleviates the tasks of creating stubs for each of the native code functions used in the managed code--effectively reducing the amount of code a developer needs to write. The method is to create a custom C++/CLI interface that takes in as inputs .NET constructs and outputs .NET constructs as well. The code within the body of the interface functions usually consists of converting the constructs into whatever native constructs the native library requires, calling the native functions with these parameters, converting the result back into .NET constructs and returning them. Using this method, I easily incorporated many of the functions of OpenCV into the platform as well as a native implementation of IRCP (which was later swapped for a C# implementation because of errors in the original C implementation).

Custom Calibration and Monitoring Web Page Interfaces

In order to quickly calibrate sensors and/or monitor the state of a *service*, separate custom web page interfaces were created for each *service*. For example, the *IMU service* has a web page interface (accessible from any modern browser) that shows the readings from each of the IMU device's axes. The technician can zero out these values and save them to a state file at the click of a button on the website. AJAX technologies, like the ones used for the *Dashboard service*, dynamically update the website in real-time. This design pattern is common across all of the real-time data sensors in the Huggable robot and is abstracted away into JavaScript and CSS libraries available for use by any *service.* Other web page

interfaces allow the technician or developer to observe *service* performance statistics that are calculated at run-time. Figure 11 show some examples of these web page interfaces.

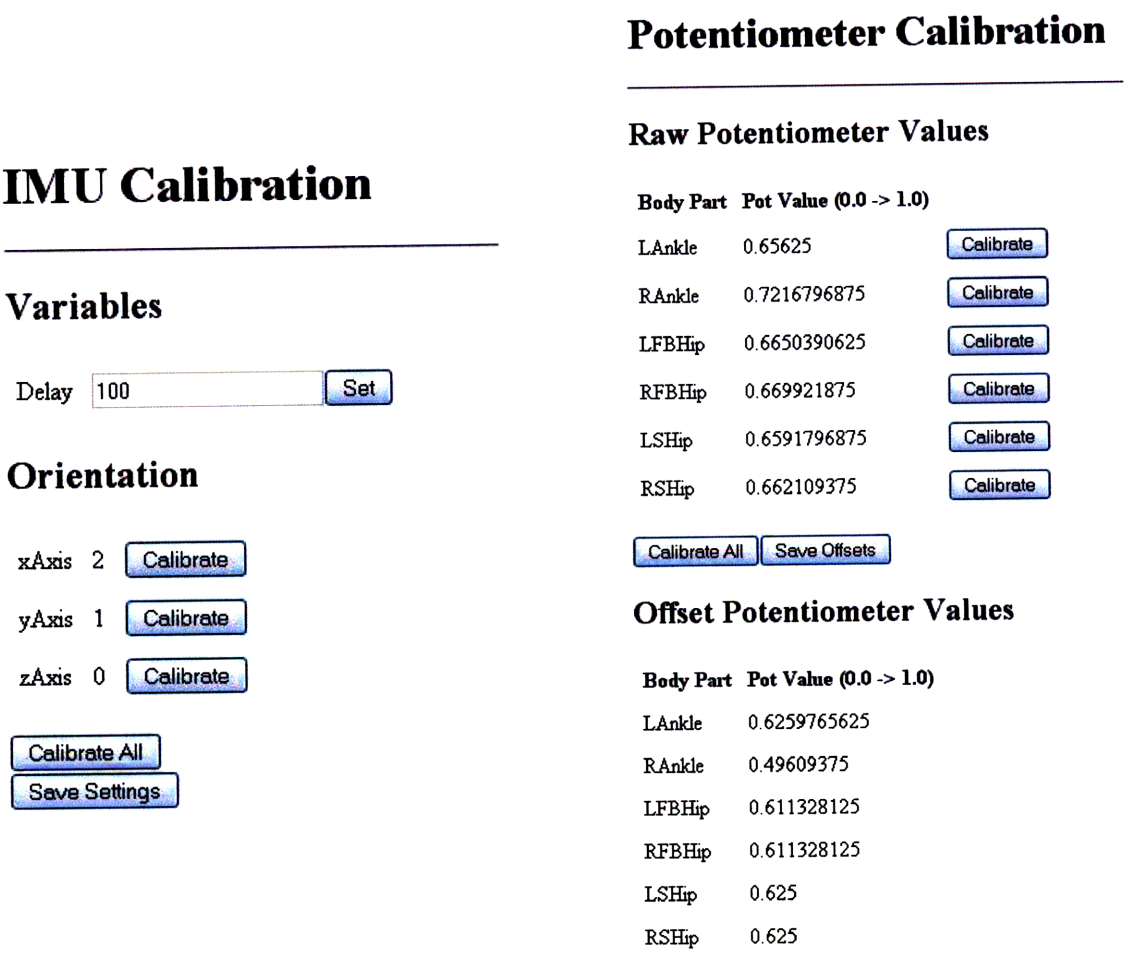


Figure 11: These screen-shots are examples of custom calibration web page interfaces. Developers can zero out sensor values at run-time at the push of a button and can persist those calculated offsets across system restarts.

The reason why the process of transmitting data to the web page interface from the *service* and sending commands to the *service* from the web page is the only thing that is abstracted away via JavaScript libraries, and not any other kind of visualization, is because it has been found in the Huggable system that sensors vary in the nature of the data that they provide. Some data are better visualized one way, while others are visualized better in another way.

Standardizing the presentation of the data would restrict the freedom to visualize the data in different ways.

There are four reasons why web pages were chosen to create the interfaces for monitoring and calibrating sensor data as opposed to the traditional desktop GUI application. One, the reason why the web has become such a popular platform for developing applications is because the programming languages used to build these client-side applications are relatively easy to learn. Typically, these web-associated languages (JavaScript and XML) are easier to become familiar with than enterprise level programming languages (Java or C#). Second, experts in these web-associated languages can specialize in developing the presentation of the sensor data and not worry about issues of operating system differences or the complexities of desktop GUI frameworks. Third, all it takes to view any one of the web page interfaces is a browser. The monitoring and calibration interface need not run on any one computer. A technician could use his or her own laptop with his or her own choice of browser to interface with any of the robot's sensors and would not have to terminal in to the computer that would be running the sensor *service*. Lastly, MSRS already provides libraries for accepting HTTP *Get* requests from web browsers and returning XML content. These reasons make it clear why web pages were chosen for monitoring and calibrating the robot's sensors.

Technologies for the Social Robotic Avatar Application

There are a series of technologies that were developed in order to satisfy the requirements of the social communication application. Again, in general, these technologies aim to provide the puppeteer with sensor feedback so that he or she can have a good understanding of their remote environment, and remove cognitive load from the puppeteer so that he or she can focus on the interaction. In this section, I divide these technologies into two types: technologies that run local to the robot, and technologies specifically

targeted for remote puppeteers.

Local Technologies

The following technologies run locally on the robot. This means that the technology runs either on the embedded computer located inside the robot or on the user's home computer (or any computer on the same *subnet* as the embedded computer). In addition, these local technologies do more than just aid the puppeteer--they can be reused for other applications besides the robotic avatar. As such, the puppeteer is not necessary in any of these technologies. However, the following technologies do provide benefits to the puppeteer such as presenting data to the puppeteer, relieving him or her of the cognitive load associated with puppeteering, and orienting the puppeteer in the remote environment.

Face Detection

The face detection technology integrated with the Huggable system helps a remote puppeteer handle the task of maintaining eye-contact with a user. The technology can detect upright and frontal faces in a video feed and denotes them by drawing squares around each face in the video feed. The robot can use the location of a face in the image to move its head so that the face appears in the center of the video feed, effectively making the robot track faces. Eye-contact is crucial in a social communication setting and this technology allows the robot to handle this social cue to free the puppeteer of cognitive load to allow him or her to concentrate more on the interaction. Figure 12 illustrates what the puppeteer sees in the video feed with this technology enabled.

Figure 12: The face detection technology helps reduce the cognitive load of the puppeteer by maintaining robot eye contact with the user.

IMU Stabilization of Video

The video stabilization technology helps to keep the puppeteer oriented in the remote environment while the robot may be being picked up and rotated about. This multi-modal technology makes use of both the camera feed and the IMU. Every video frame from the video camera of the robot is coupled with the roll position of the robot given by the on-board IMU. The video frame is then rotated by the negative of the roll value. So if, for example, the robot is rolled 20 degrees, the video frame captured at that moment would be counter rotated by -20 degrees. This counter rotation has the effect of keeping objects upright in the video frame instead of being rotated with the robot. Figure 13 shows a time-sequence of this technique. This idea was modeled after the fact that while humans may tilt their heads at different angles, and hence tilt what they see, their brains allow them to realize that what they see is not actually tilted. This technology also helps other vision dependent technologies function, such as the face detector. One current limitation in the face detector is that it only detects faces that are portrait and upright with respect to the image. Faces that are tilted or in profile are less likely to be detected. Similarly, if the robot was placed on its side, no faces would be detected unless the faces were tilted in the same

orientation of the robot. However, the robot is able to detect faces in this kind of image because of the multi-modal use of the robot's camera and IMU sensors.

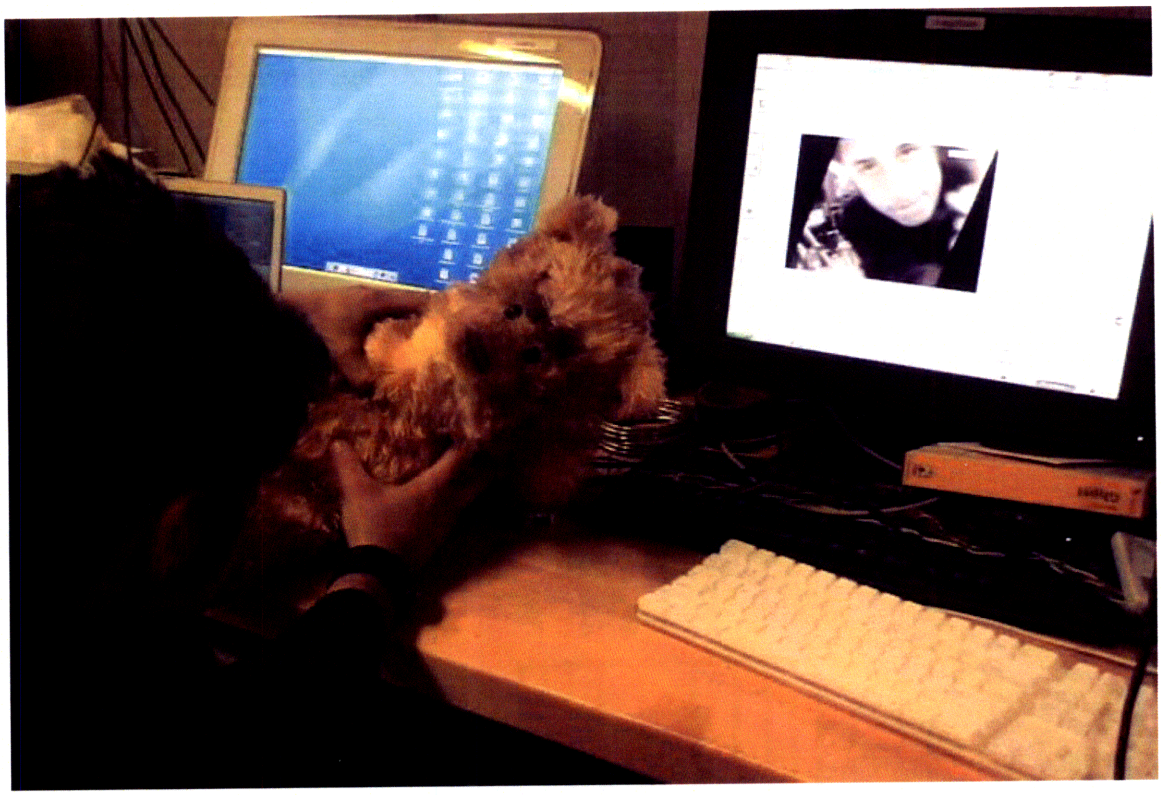


Figure 13: When the robot is tilted on its side, the reading from the IMU device is used to counter-rotate the incoming video stream such that the spatial orientation of objects in the video remains constant.

3D Virtual Robot Model

The C5M behavior system features a 3D virtual model of the robot. Potentiometers in each of the robot's degrees of freedom broadcast their current position and subsequently update the joints of the virtual model. By virtue of this coupling, moving a joint on the real robot moves the joint of the virtual one, and vice-versa. This potentially allows for the puppeteer to easily see how much range of motion is left as they move the robot's joints. It also provides a virtual mirror to what the user sees as they look at the robot. Besides visualizing the positions of each of the actuated joints of the robot, the virtual model also displays the joint positions of the passive limbs such as the legs and feet. It also visualizes the data from

the skin sensors of the robot by highlighting portions of the 3D model's surface texture and the data from the IMU by rotating the virtual robot. Figure 14 shows an example of the model. Because this 3D model is rich in information and illustrates sensor data in a graphical and spacial representation, it is sent to the remote puppeteer to aid him or her in understanding how the robot is being interacted with.

Figure 14: This is a screenshot of the virtual 3D model of the robot. In addition to showing the orientations of each of the robot's DOFs, the 3D model can also serve as a display of spatial data such as where the robot is being touched via the skin sensors, or the orientation of the robot with respect to the ground by rotating the model appropriately.

IMU Motion Classification

In the case of the data from the IMU sensor, visualizing the raw data from the sensor on the 3D virtual model (as described above) does not convey all of the information that would be relevant for this social communication application. For example, if the robot is being bounced up and down, while not changing its orientation with respect to gravity, the 3D model would not be able to illustrate this to the puppeteer and neither would looking at the video feed, unless the puppeteer pays very careful attention to the way the incoming video

changes, but this would potentially not allow the puppeteer to concentrate on the conversation with the user. This is remedied by a technology that runs locally on the robot to classify how the robot is being moved. Algorithms that process the data from the IMU are capable of identifying whether the robot is being picked up, bounced, or rocked [25]. These algorithms are based on training a neural network on IMU data and using it to classify the motion. Presenting this processed information in the form of a simple cartoon animation greatly aids the puppeteer in identifying how the robot is being interacted with, with respect to motion. This is because it is easier for a user to recognize an idea by seeing it rather than making a symbolic jump from a word to an idea via recall [18]. Figure 15 shows some examples of the graphical representations of the robot motion that are presented to the puppeteer.

Figure 15: These stylized cartoons of the robot's detected motion are easier to recognize than tracking the motion in the 3D model or reading a keyword description of the motion. The motions shown here (from left to right) are *no motion*, *bouncing*, *pick up*, *rocking*, and *over stimulated*. Animations courtesy of Lily Liu.

Skin Technology

While the full body sensate skin has not yet been completed and integrated with the robot, some of the preliminary visualization work has been done and can be provided to the puppeteer. The skin technology is designed to make use of hundreds of electric sensors that cover the entire exterior of the robot. Each sensor is capable of sensing touch, and high-level algorithms are capable of detecting how the robot is being touched (tickled, grabbed, slapped, etc.). More information on the skin technology can be found in [26]. An MSRS *service* called *Skin* and a corresponding web page interface were created for this

technology. The web page displays a cartoon image of the robot and different parts of the cartoon are highlighted when the user touches the robot. The web page interface makes use of the same AJAX technologies developed for other *services* and can provide the puppeteer with an idea of where the robot is being touched. Just like the IMU, providing the puppeteer with just a cartoon image of where the robot is being touched might not be enough information for the puppeteer to fully understand how the robot is being interacted with. A solution to this problem is discussed in the Future Improvement section. Figure 16 shows the preliminary implementation of visualizing the touch data from the skin sensors.

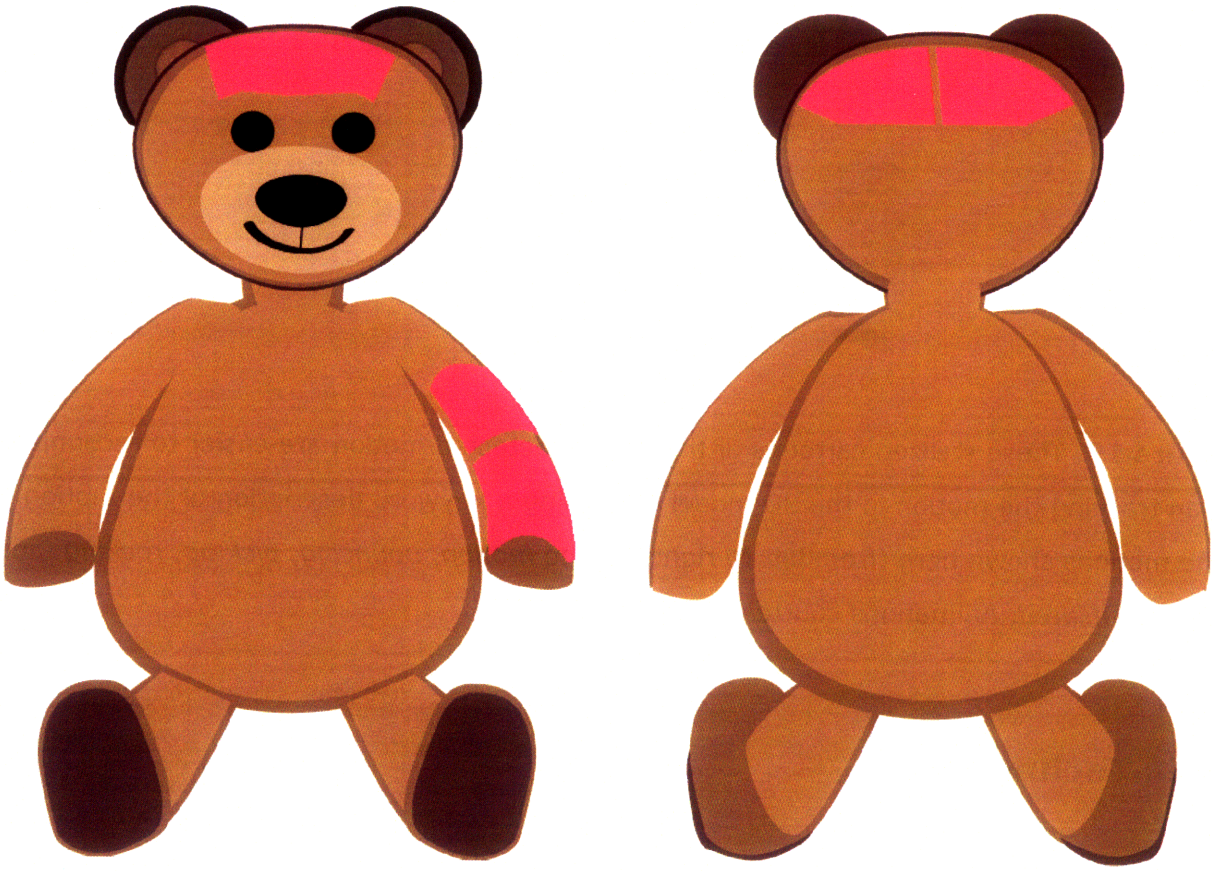


Figure 16: A prototype visualization of how the robot is being touched. Contiguous pink regions represent an activated touch sensor on the robot. Touch classifications, such as touch, pet, or tickle, are not represented here. From the visualization, the puppeteer might speculate that the robot is being held by the arm while patted on the head. Drawings courtesy of Heather Knight.

Remote Technologies

The next set of technologies exist on the puppeteer's side of the interaction. Like the technologies that run on the local side of the interaction, these technologies serve to present data to the puppeteer, relieve him or her of the cognitive load associated with puppeteering an eight DOF robot, and orient the puppeteer in the remote environment.

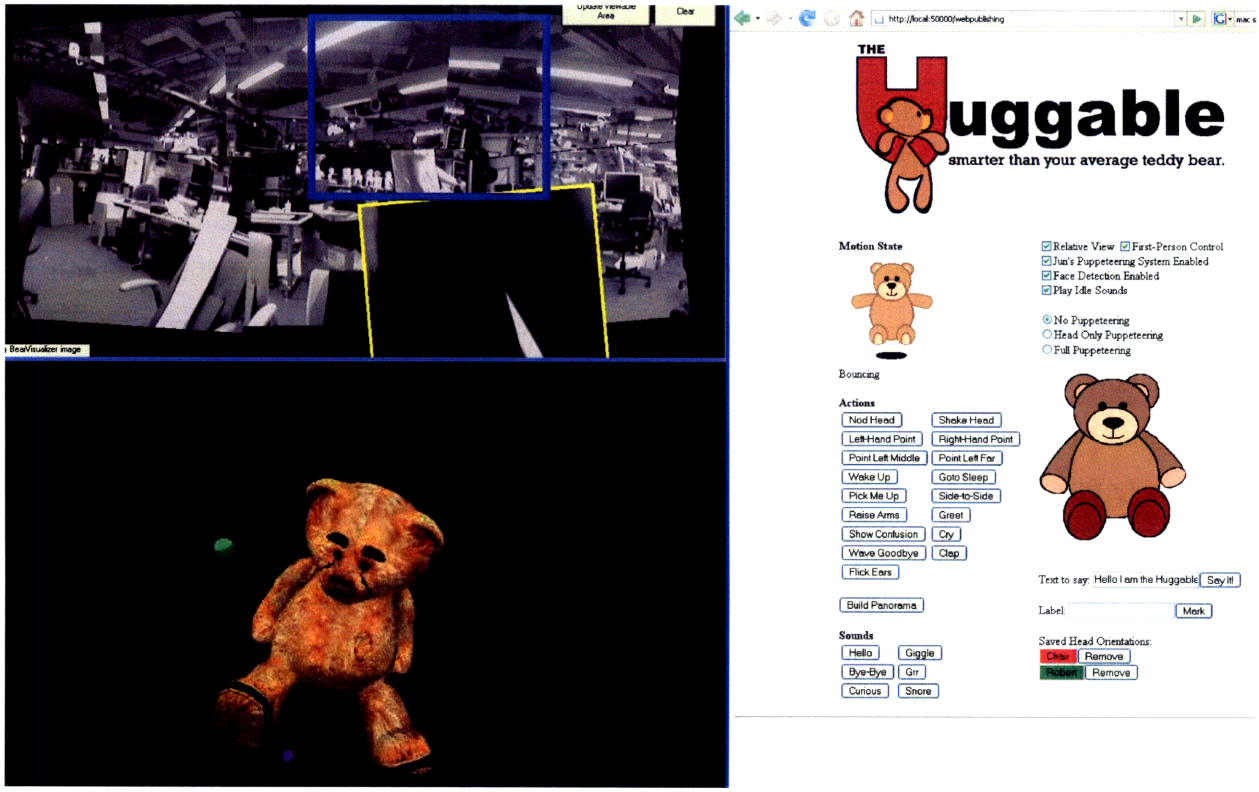


Figure 17: A screenshot of the puppeteer's interface. This interface contains three main components: the stale panorama (top left), the 3d virtual model (bottom left), and the web interface (right).

Stale Panorama

One phenomenon of teleoperation is the fact that the robot's video feed presents the puppeteer with *tunnel vision*. Humans not only depend on their high-resolution fovea for

localizing themselves in an environment, but also on their peripheral vision. It is more difficult to localize oneself if the only data is from the fovea, or in our puppeteer's case, a 320x240 video feed. One method to cope with this phenomenon is to use a camera with a wide viewing angle (or a "fish-eye" lens). However, this type of camera could then not be used for face detection nor some forms of motion detection because of the distorted nature of the video image. I considered using two cameras (one in each eye of the robot)--one for sending video to the user (the wide view angle camera) and the other for capturing video for the video dependent services (the regular camera). This had three drawbacks: it increased power consumption due to the addition of the second camera, it increased the cost of the robot, and it complicated the mapping between locations in the regular camera feed and in the wide angle camera feed. Instead, I chose to implement what I call the stale panorama, shown in figure 18. This involves the robot autonomously looking around the room, capturing video frames and storing them with the associated position of the robot's head. As these frames are collected, they are superimposed on a much larger canvas at a location corresponding to the position of the robot's head at the time the frame was captured. The robot chooses to look at parts of the environment that it has not seen yet and eventually fills in the entire canvas. The result is a collage of still images that are positioned so that they build a stale panorama of the environment. The only non-stale part of the canvas is the current position of the robot's head, which is instead a streaming video feed. The completed stale panorama is then used as a pointing device, enabling the puppeteer to click on a point in the panorama and causing the robot to autonomously handle looking in that direction.

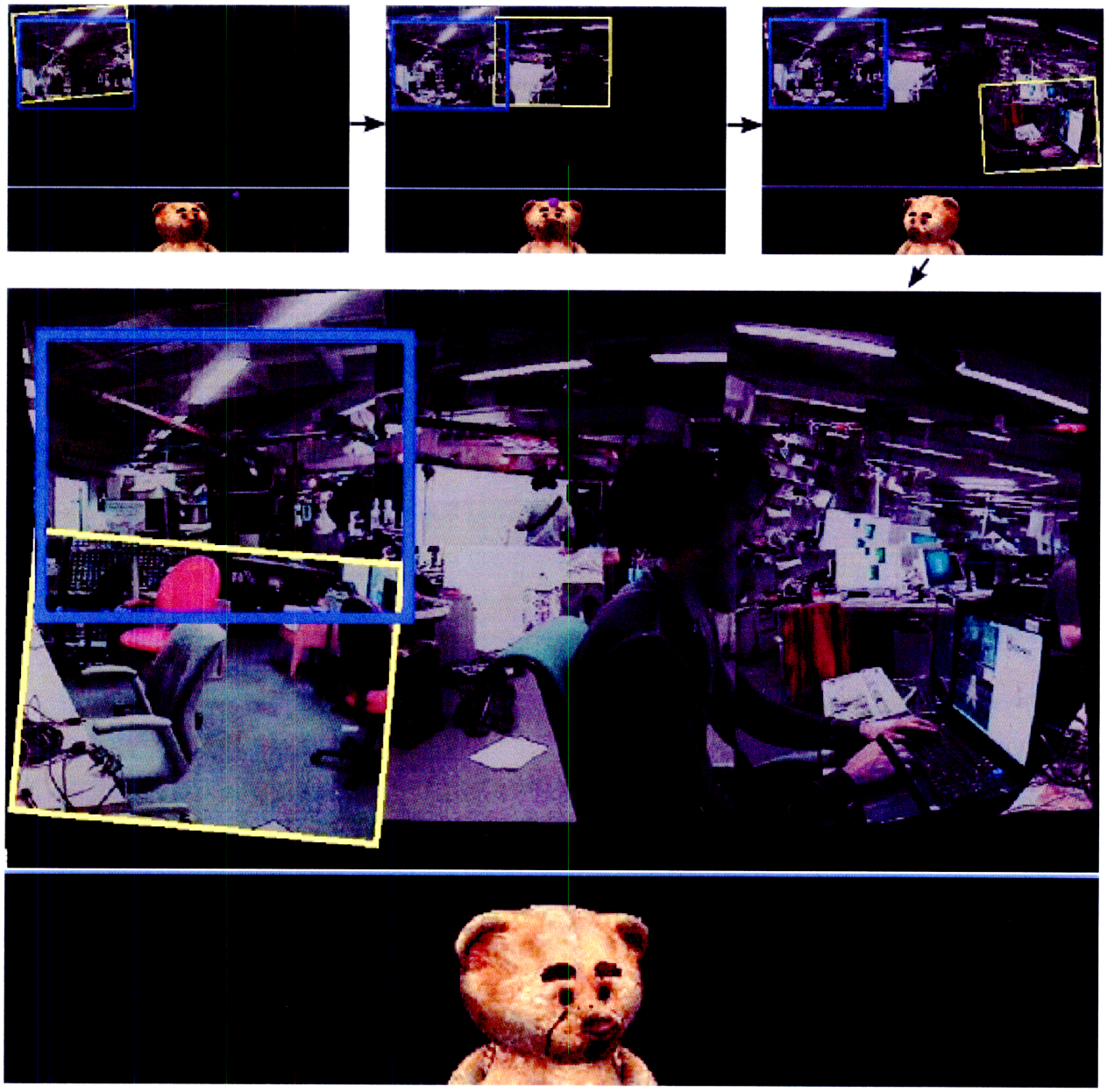


Figure 18: A time-sequence of the process of building the stale panorama. The blue box represents where the puppeteer would like the robot to look and the yellow box indicates where the robot is currently looking. During the building of the stale panorama, the robot ignores any gaze direction requests, hence the discrepancy between the blue and yellow boxes. Note that simple affine transformations of the video frames based on the position of the neck is a rough attempt to build a panorama of the remote environment.

The stale panorama technology is no longer relevant if the scene in the panorama drastically changes (i.e. the robot becomes surrounded by people) or if the robot is picked up and moved. The way I cope with the former is by continually updating the stale panorama with frames captured while the robot's head is not in motion during some time-interval. While in some cases the stale panorama might not be very accurate in reflecting what is in the remote space for constantly changing environments, it does work well for relatively static environments, such as a bedroom. The above technique fails to solve the relevance problem in the case where the robot is picked up and moved. It fails to work in this case since while the stale panorama is being updated constantly, the rest of the panorama would be completely incorrect (assuming the robot has been placed in a new environment or is now facing a different direction). The solution to this problem is to completely remove the existing stale panorama when the robot is picked up or moved and only start building a new one when the robot has been placed back on a stable surface. The IMU sensor offers us the classification of these types of movements. What the puppeteer will end up seeing is when the robot is picked up and is in motion, the live video feed is zoomed in such that it takes up the entire panorama window. When the IMU detects that the robot is no longer in motion, the live video feed is zoomed out and the panorama no longer exists, but instead is just filled with a black color. The robot then begins to build the stale panorama as before. Figure 19 shows a time sequence of this process.

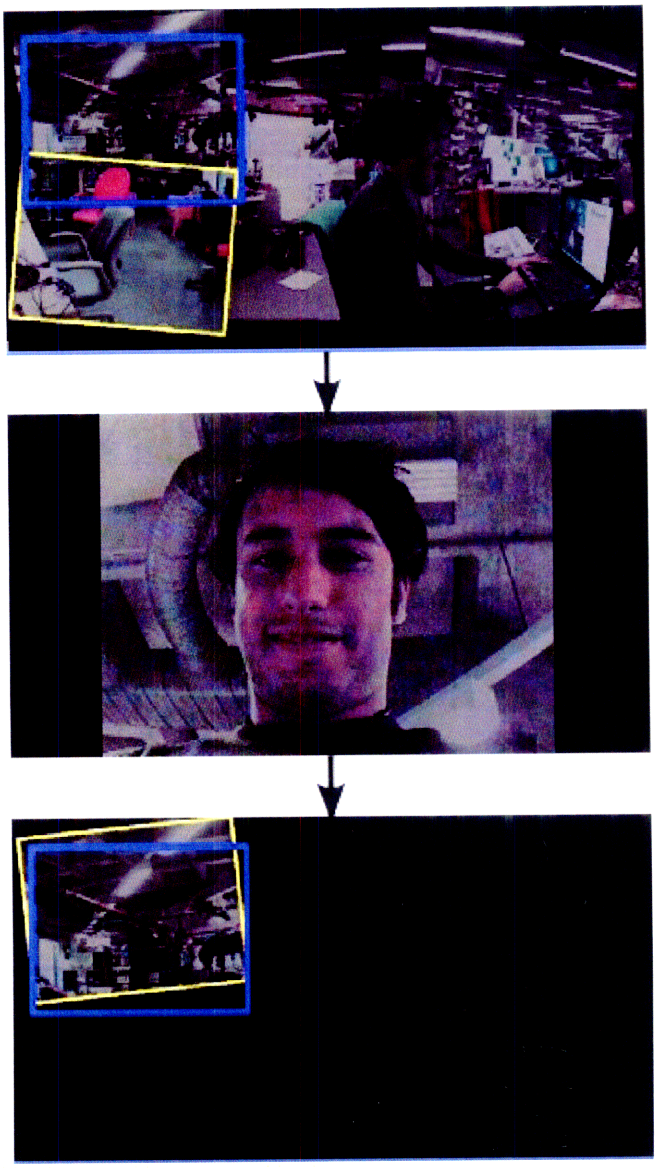


Figure 19: A time-sequence of what happens when the robot is picked up and moved to a new location. The first image illustrates the stale panorama before the robot has been picked up, the second is when the robot is in motion (i.e. being carried), and the third image illustrates what the panorama looks like when no motion is detected by the IMU. In the third image, the stale panorama is cleared out since the old panorama no longer reflects the new remote environment. A new stale panorama would then be subsequently built.

From figure 18, we can see that simple affine transformations of the video frames according to the robot's neck position at the time the frame was taken results in inconsistencies at the edges of these images. These inconsistencies can hinder the intended effect of orienting the puppeteer in the remote environment. Crystal Chao (a graduate of M.I.T.) has begun testing stitching algorithms to accurately transform these images so that they align much better and improve the overall clarity of the stale panorama. She uses an implementation of the SIFT algorithm [17] to identify scale-invariant *keypoints* in the video frames, and then applies the RANSAC algorithm [9] to find a non-affine transformation matrix. She then uses OpenCV's algorithm for warping images to apply the non-affine transforms to the frames in

the panorama. Figure 20 illustrates this technique for improving the clarity of the stale panorama technology. The process for stitching together two images takes about 15 seconds which means that this technique can not be used in real-time. Instead, Images are collected in the background, and the panorama eventually is constructed and presented to the puppeteer as an asynchronous task. One way to improve the running time of the algorithm would be to use the neck joint potentiometer data of the robot to help the RANSAC algorithm find the best match between two images. It is important to note here that the stitching aspect of the stale panorama is being developed independently of the Huggable project and will be integrated when we develop our own versions of the RANSAC and SIFT algorithms, because we have filed several patents for the Huggable platform before the inclusion of the these copyrighted algorithms.

Figure 20: This figure illustrates the image stitching process. The top two images are the inputs to the process and the bottom image is the result. The red dots indicate the input images' best matching SIFT features found via RANSAC. The resulting non-affine transform

then is applied on the right input image and superimposed on the left input image.

The stale panorama technology helps the puppeteer in three ways. One, it alleviates the burden of having to manually control the individual motors of the neck joints in order to look around the remote environment by providing a point and click interface. Two, by removing this burden, the robot also helps with issues of latency across the internet since it is cheaper in time to send just one message containing coordinates of where the puppeteer would like the robot's head to be pointing at, instead of sending individual move commands which could result in overshooting the visual target due to latency. And finally, it gives the puppeteer a better understanding of the remote environment by removing the *tunnel vision* problem.

Object Labeling

A technology was created to aid the puppeteer in the task of quickly moving the robot's head to look at various objects or people in the remote environment. This technology is especially useful if the puppeteer is talking to several people through the robot or if the puppeteer needs to repeatedly shift the robot's vision between a person and a open book. The way it works is that the puppeteer can "take a snapshot" of the current neck position of the robot and associate a label with it. The puppeteer can then recall the position of the robot's head at the click of a button. Currently, the interface to this technology is provided on the main Huggable web page interface described below.

Web Interface for Puppeteering

The *WebPublishing service* within the Huggable software system acts as a control panel to the robot for the remote puppeteer by providing a webserver on which to serve HTML, JavaScript, CSS, and other web-related content. This web page, supplemented with AJAX technologies, presents information to the puppeteer as well as provides controls for initiating various robot behaviors. Figure 21 presents a screen-shot of this interface while it is in use. The interface provides buttons for sending commands to the C5M behavior system

to play back various animations. These animations include waving, wiggling of the ears, nodding, etc. A puppeteer can use these prerecorded animations to make the robot more engaging through its life like motions. In turn, this can enhance the interaction between puppeteer and user [15]. It also has buttons to play prerecorded sounds on the robot (e.g. cute bear noises) and a text field to send a phrase to the Cereproc text-to-speech engine. As mentioned before, the web interface also provides some controls on adding or removing object labels. Figure 21 shows an example of what these labels would like to the puppeteer. The web interface also allows the puppeteer to change how much autonomy he or she wishes the robot to have. In Huggable system, there are currently only three degrees of autonomy and they refer only to motor control. They are: *no puppeteering*, *head only puppeteering*, and *full puppeteering*. There are check boxes that allow the puppeteer to enable or disable other technologies in the Huggable system such as face detection, IMU stabilization of video (shown as "Relative View" in figure 21), the embodied puppeteering system (shown as "Jun's Puppeteering System Enabled"), and the of playing sound effects when the robot is idle.

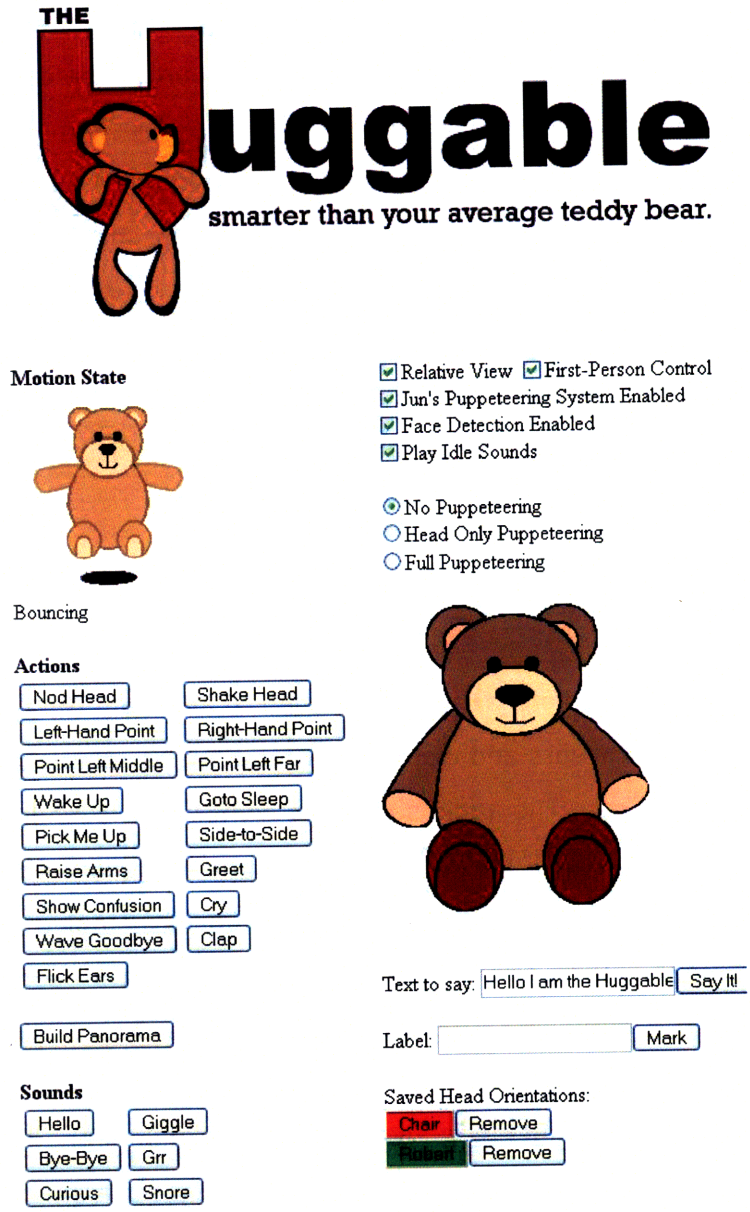


Figure 21: This is the web page interface that is accessible to the puppeteer from a web browser. The interface provides controls for play back of prerecorded animations, prerecorded sounds, custom speech, and complex behaviors such as building the stale panorama. It also shows processed data coming from the robot's sensors such as the IMU (shown as the bouncing cartoon) and the joint potentiometers (shown as the left and right feet of the right-most cartoon highlighted in red).

The web page interface also serves as a conduit for data incoming from the robot. One goal of this interface was to make sure that the data coming in would not be too complex, but instead, easy to recognize. Since most of the puppeteer's attention is spent looking at the video feed, he or she might have little remaining cognitive attention to monitor how the robot is being moved or touched. The website currently displays the motion classification detected by the IMU device, as well as a cartoon doll showing which limbs are currently being activated via monitoring the of potentiometers in each of the robot's joints.

Searching for the right animation or sound to play in the sea of prerecorded items can take an inconvenient amount of time. This search time can greatly interfere with the ongoing interaction between puppeteer and user. For example, if the puppeteer wants to convey affirmation by making the robot nod its head, the time that it would take for the puppeteer to find the button and press it might not be short enough for the user to associate the affirmation with some event that had just happened. I believe that the way the interface is layed out is currently unusable for a sustained interaction as made evident in the evaluation section of this paper. In the Improvements to the Huggable System section, I discuss new ways of allowing the puppeteer to control the robot's behavior.

Audio Chatting

A duplex audio channel is streamed to and from the puppeteer to the robot. Microphones in the robot's ears and a speaker located in the robot's snout record and play audio, respectively. The audio streaming component is modularized into MSRS *services* which allows for the reuse of code on both the embedded computer and the puppeteer's computer. The audio that the microphones pick up are sampled at 22,050Hz. The frequencies in human speech are rarely above 10kHz, except for some fricatives such as the "S" sound [30]. A sampling rate of double 10kHz was chosen to account for aliasing effects. The lowest possible sampling rate was chosen in order to reduce the latency between puppeteer and user while maintaining enough clarity to understand the spoken word. Each sample is 8 bits in resolution and only one channel is sampled. A small audio buffer size of 500 bytes was chosen to also minimize the amount of latency between puppeteer and robot.

Embodied Puppeteering

Jun Ki Lee, also a researcher in the Personal Robotics Group at the MIT Media Lab, has developed an embodied puppeteering system. This is a wearable set of sensors that detect movement and orientation of the arms and head in order to control an avatar [16]. Figure 22 illustrates what kind of sensors are worn by the puppeteer. I helped integrate his system

along side the rest of the MSRS *services* in order to provide the puppeteer with another mode of puppeteering. His system currently allows a puppeteer to control an avatar in two ways: via direct manipulation or via gesture recognition. The direct manipulation method streams the data captured from the worn accelerometers and magnetometers to the C5M behavior system. These data are then translated into motor commands for the robot. The end result is when the puppeteer moves his or her arm, the robot's subsequently moves its arm. While this method is a natural way to control the robotic avatar, the latency and throughput restrictions that are imposed by the internet might make it impossible to control the robot via this method.

Figure 22: The wearable puppeteering interface includes worn accelerometers and magnetometers at the head and elbow joints as well as the hand-held Wii remote and Nunchacku. The development of this interface is not a part of this thesis.

The second method, gesture recognition, allows the puppeteer to make deliberate gestures to invoke animations to be played back on the robot. For example, if the puppeteer makes a waving gesture, his system recognizes this gesture and sends it to the C5M behavior system where it is played back on the robot. Therefore, there is a one-to-one correspondence between the gestures that the puppeteer makes, and the prerecorded animations provided by the robot. While sending only one message with the type of gesture to be invoked on the

robot is more robust with regards to the internet, the actual classification of the gesture is difficult and the gesture is only recognized once it has been completed, hence injecting another source of delay between when the puppeteer intends to act, and when the robot acts. In either case, we are looking forward to performing user eva luations comparing the web page interface and the embodied puppeteering system. The best system will most likely be a combination of both interfaces.

Evaluation

A lot of work went into building the technologies to solve many of the engineering problems associated with building a software platform for a semi-autonomous robotic avatar for the social communication application. Unfortunately, no full-scale user study could be conducted due to time constraints but there was enough time for some basic performance measurements and an informal user trial.

Puppeteering Related Performance Statistics

In order to gain an understanding of the performance of the system and where performance improvements might be best suited, I instrumented certain *services* within the Huggable platform with performance statistics gathering code. The performance results of some of these *services* are actually output in real-time to their respective web page interfaces. For example, the *Gramps service's* web page interface provides a real-time reading of the frame-rates of the video from the virtual 3D model, and of the video camera feed. The following statistics were gathered during a typical usage scenario of the robot (all systems on, moving the robot's limbs, playing sounds, and speaking through the microphones). All but one of the computers involved in the Huggable system were connected via ethernet on a closed private network. The MacBook Pro laptop was the only computer connected wirelessly to the private network.

Variable	Value
Framerate of video from robot's camera	10.3 frames per second
Framerate of video from virtual 3D model	8.1 frames per second
Face detection rate (when face is present)	2.7 frames per second
Audio latency	0.45 seconds
Robot animation latency*	1-5 seconds
Text-to-speech latency	2.5 seconds
IMU motion classification	0.9 - 2 seconds (assuming true positive)
Joint potentiometer activation latency**	0.2 seconds

* the time it takes from when the puppeteer presses a button on the web interface to when the robot physically performs the animation

** the time it takes from when the user wiggles the foot of the robot to when the puppeteer is notified via the web interface

From my research group's own informal usage of the system and in the informal user trial decribed below, many of these values are reasonable in that they satisfy the six requirements outlined in the Problem Statement for the social communication application. However, the statistics that inhibit a smooth interaction between puppeteer and user through the robot include the audio latency and the robot animation latency. While the audio comes out clear on either end, there is the problem that the puppeteer can hear him or herself when he or she speaks, due to feedback from the microphones on the robot. Humans hear themselves all the time when speaking, but when a delay is introduced between when something is spoken and when it is heard by the same person, it can be confusing and difficult to continue speaking. Since the latency for the streaming audio is 0.45 seconds, the delay is apparent and there were complaints of difficulty and confusion while speaking through the puppeteering interface.

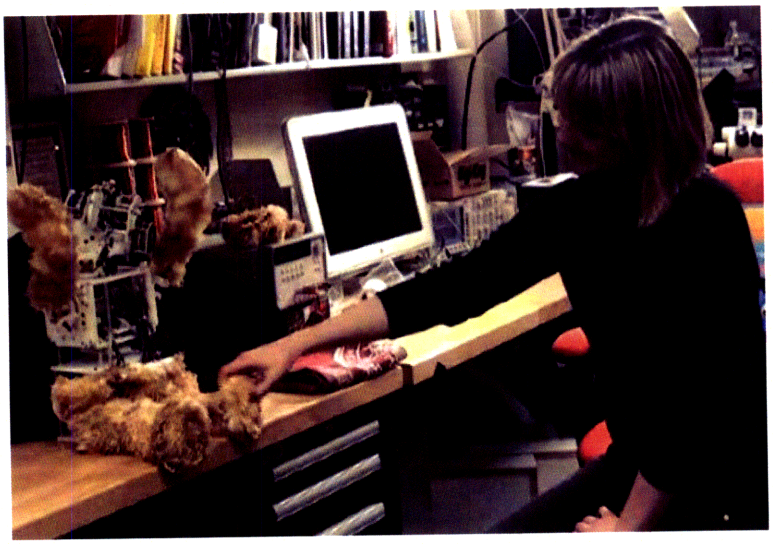
While the audio echo problem can be tolerated by some puppeteers, the latency in the robot animation can really hinder social communication. In the cases where the puppeteer

intended to nod the robot's head in affirmation to the user, the latency was so great that sometimes the user would repeat the question or the puppeteer would think that their initial command did not go through and hit the button a second time. In the current implementation of the Huggable system, repeated animation button clicks results in repeated animation playbacks, which means that the robot would execute two nods, when the nod animation was clicked twice. The problem with this is that in the current implementation of C5M, an animation cannot be interrupted, so a queued up series of animations would render the puppeteer without control of the robot until all animations completed.

Another performance related drawback is the disparity in latency between the motor joint positions and the video feed. This is most noticable in the stale panorama. The incoming video feed is placed at a location on the stale panorama that corresponds to the position of the robot's neck, if these data are not synchronized, the panoramic effect is lost. The software system developed for Stanley uses a time-stamping mechanism to synchronize data from all of the sensors of its autonomous vehicle. This would be one way to establish synchronization for the Huggable system since it does not already have this feature.

Informal User Trial

In April 2008, my research group filmed a video demonstrating the capabilities of the robot. A script was written in which I puppeteered the robot to interact with a few of my lab mates while participating in various activities ranging from reading a book together to coloring a drawing together. The robot was situated at one end in our lab and I was seated in a closed office across the lab where I could not hear nor see the robot or the user. This experience proved to be a valuable informal evaluation of the puppeteering interface.



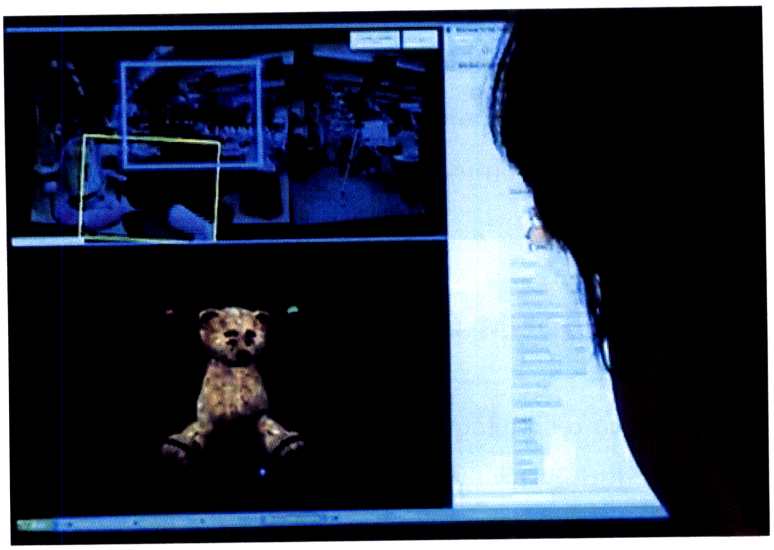


Figure 23: A scene from the demonstration video my research group filmed of the Huggable robot. The top image is of the user interacting with the robot. The bottom image is of the puppeteer and his interface.

Web Interface Usage

The first thing I noticed when trying to do a simple greet animation on the robot was the fact that there are so many available animations. Reading each button took some time, and since there are so many animations, their button size must be kept small to leave enough screen real-estate for other interface elements. Fitts's law shows that it will also take a long time just to click any of the buttons because of their small size [31]. Perhaps fewer options should be presented to the puppeteer so that the buttons could be bigger and possibly

labeled with an illustration of the animation instead of text. Another idea might be to use a touch-based device to select the animation--this would remove the minimum mouse cursor travel time mandated by Fitts's law. Also, since my attention was mainly focused on the video feed, and hence was the location of my mouse cursor, it took some time moving my mouse cursor from the stale panorama over to the web interface where the buttons were. In some cases, the opportunity to respond to the user with a pre-recorded sound in the context of the interaction was missed due to this delay.

One surprising thing I encountered was my preference for using the stale panorama over the object labeling technology for looking back and forth between the user and an open book. This preference may be due to the unwanted initial effort required to label each of the head poses. This effort, however, is not applicable in the stale panorama's case. Another reason might have been that the buttons to move the robot's head to a given label were located on the web interface, to the right of the stale panorama. Again, this distance to travel with the mouse cursor would hinder the social communication.

Stale Panorama and 3D Virtual Robot Model

I came to exclusively rely on the stale panorama technology for controlling the robot's gaze. Consequently, I spent about 90% of my time during the filming looking at the stale panorama, and rarely glancing at the 3D virtual model or the web interface. I believe that the stale panorama was essential for orienting myself in the remote environment, and thus I was extremely dependent on it for locating objects in the remote environment (such as finding a book to read) and maintaining the robot's gaze (maintaining eye contact, shifting the robot's gaze from the book to the user and back). Since most of my attention was on the stale panorama, I could have potentially missed data coming from the robot's other sensors, such as the joint potentiometers or the IMU. One suggestion might be to superimpose a "heads up display" (HUD) on the stale panorama. This would involve superimposing some of the elements presented on the website, around the borders of the stale panorama display. The HUD has been a very successful user interface element in first person shooter computer and video games, in which the user's attention is highly dependent

on vision in the virtual environment (or in my case, the remote physical environment).

When I did look at interface elements that were not the stale panorama, it was usually for a specific and reoccurring purpose. For example, when I wanted to say, "Hi so-and-so!", while making the robot wave, I had to monitor the 3D virtual robot video feed for signs that the robot started to wave so that I could time my speech accordingly. The robot animation latency, discussed in the Puppeteer Related Performance Statistics section, made this timing process even more difficult. Another example of not looking at the stale panorama was when the robot was picked up by the user. Since I recognized from the video feed that the user was approaching and picked the robot up, I looked to the web interface to see what kind of motion was detected via the IMU motion classification technology.

Puppeteering and the Sympathetic Interface

I came to notice that what I was physically doing in the office was not the same as what the robot was doing. For example, when the robot waved, I was not also waving. This is due to some obvious reasons. One, my hands were occupied puppeteering the robot and so I could not wave. Two, I knew that my gestures were invisible to the user and felt no need to wave. And third, the interaction was scripted and so the gesture was not as spontaneous as it would normally be. However, this was still the case when I made the robot perform a subtler animation such has nodding its head. It was my initial hypothesis that the puppeteer would feel the need to perform some of the gestures that he or she was making the robot perform, since they were controlling an extension of themselves, but this did not happen with me. I believe that this gestural disconnect led to the problem that the robot was not being lively enough during filming. On most occasions, I forgot to keep the robot's ears flicking in response to comments, or to nod the robot's head in addition to saying, "yes" or "OK".

This gestural disconnect problem might be solved by Lee's embodied puppeteering interface since the gestures that the puppeteer physically performs will be sent to the robot (see Embodied Puppeteering in the Technologies for the Social Robotic Avatar Application

section). This might result in the robot being more life like. But another idea would be to use a sympathetic interface robot (SIR) for controlling the Huggable robot's gaze and gestures. The SIR would consist of a robot which contained only potentiometers in each of the joints to detect and broadcast joint positions. These positions could then be sent to the Huggable robot for processing. So for example, if the head of the SIR was moved to the left, the Huggable robot's head would move to the left, or if the SIR's arm was moved in a waving gesture, the gesture would be detected by a classifier and the Huggable robot would invoke the waving animation. Using the SIR, I believe that the puppeteer will be more aware of the robot's idle gesturing and will be able to more naturally control the robot. One drawback of Lee's puppeteering system is that if the gaze of the robot is controlled by movement of the pupeteer's head (via an accelerometer located in a hat that the puppeteer wears on his or her head), then when the puppeteer wants to move the robot's gaze to the left, the puppeteer might no longer be able to see the screen, since his or her own head would be pointed to the left, away from the computer screen. Use of a SIR would allow enough gestural disconnect to perform non-embodied actions such as directing robot gaze, but not so much so as to overwhelm the puppeteer with the complexity of a completely non-embodied puppeteering interface such as the website interface.

A Testament to Telepresence

Perhaps the most interesting result of filming this interaction demonstration happened when I was not trying to puppeteer the robot for the purposes of the film. In between takes, the director of the film began to speak to the actors about what needed to be changed and which parts went well. When he began to speak to me, he looked at the robot in the eyes and spoke to it as if it were me. He did not have to look at the robot when he spoke, he could have just as easily spoke facing any other direction since the robot's microphone was of good quality. He gestured at objects in the same space and he even nodded his head at me for confirmation that I understood his directions.

We agreed later that the fact that I moved the robot's gaze back and forth between speakers (the actor and the director) solicited the need for eye contact when speaking to

the robot. In addition, since I moved the robot's gaze to look at whatever they were pointing at convinced them that I understood the physicality of my remote environment. When these factors became commonplace, telepresence was working in its truest form.

Future Formal User Testing

In the the future, a formal user trial could be conducted by allowing certain groups of users to use the puppeteering interface in two types of settings. The groups of people that I would like to target are elerly people--those who haven't had much experience with computers or the internet, age 50 or older; adults--over the age of 30 who may not be as familiar as the younger generation with the latest web technologies and the associated understanding of those technologies; and finally, expert users--those who will be puppeteering a robot as their full-time job. My interest in the last group is to verify that the puppeteering interface is efficient to use by someone who is fully-trained in all of the robot's capabilities.

The two types of settings that I would like to target are to have these puppeteers attempt to read a story with one of two people. First they will read a story to a child actor who is trained in the interaction and then read a story to a child that the puppeteer agrees to bring in themselves. The two different settings ensure that the puppeteer is able to use the puppeteering interface at all with someone who is patient and that the puppeteer is able to interact with a user who has never interacted with the Huggable robot before.

A standard user test will be conducting that will include filming of both the puppeteer and user, data recording of all the data captured by the robot (including video, audio, and actions), and note taking. There will be a brief introductory session that will include an explanation of the puppeteering interface and the robot's capabilities (except in the expert's case where this training session will be much longer and contain some practice runs). The task that the puppeteer is instructed to perform will be step-by-step and he or she will be requested to fill out a survey at the end to assess how visceral the interface is, how much they understood their remote environment, how well they understood how the robot was being interacted with, and other qualitative questions on the interaction experience. There

would be an analogous survey for the users that their respective puppeteers bring in. The results from the study will be used to improve the puppeteering interface's existing technologies as well as add new ones (and possibly new sensors) to address the needs presented by the user trial.

Improvements to the Huggable System

Just as the work in this thesis was divided between work done to develop an extensible platform and work done to create the necessary components for the social communication application, the suggested improvements to the Huggable system will be so divided.

Improving the Framework

When my research group begins to conduct user studies and user testing with the Huggable robot, they will need a logging mechanism. This *service* would go beyond the logging service provided by MSRS. It would provide a clean API for logging a variety of data types. Each log entry would be timestamped not only with the current time, but perhaps also with session data such as which aspects of the applications were being tested during a given user study, etc. The logging *service* would be as invisible as possible to the rest of the Huggable platform so as to make it easy for developers to build future MSRS *services*. Ideally, it would be mediated through the HuggableServiceBase class and all of its logged data would be stored in an ACID [29] type database instead of a simple XML file. Finally, it would also be able to accept logging requests from systems outside of MSRS, such as any system that supports IRCP.

On many occasions during development, my group spent an unnecessary amount of time tracking down hardware malfunctions. While the custom calibration web interfaces for many of the *services* do display some low-level sensor readings to help a technician deduce a hardware problem, this mode of display truncates the temporal part of the data. In other words, we cannot examine how the sensor misbehaves over time. One improvement to

remedy this would be to create an API that allows for the real-time drawing of dynamic graphs via JavaScript and HTML. The reason why this improvement should use web technologies is so that we can leverage the advantages of having a web-based interface to the diagnostic pages for the various Huggable sensors. These advantages were enumerated in the Custom Calibration and Monitoring Web Page Interfaces section.

When the developers remove the robot's tether, close the robot up, and begin running the robot on battery power, power consumption and heating will become issues that will have to be addressed. An improvement to the Huggable platform would be the addition of another *Dashboard*-like *service* that would monitor the "health" of the robot's electrical components. This type of service would be invaluable to diagnosing the early engineering problems associated with moving to an untethered robot.

One issue that will become very apparent once this platform is ready to be tested in people's homes is security. One of the reasons that the MSRS platform was chosen over others was for its security capabilities. MSRS allows a developer to restrict which messages certain *services* can send and the messages' associated authentication levels. While this security capability has not been utilized for this thesis, it is absolutely necessary if this platform needs to be able to transmit sensitive data across unsecured channels. It would be worse if someone other than an authorized person could gain control of the robot remotely.

Improving the Social Avatar

It was apparent during the informal testing of the robot that switching between levels of autonomy was too cumbersome a task to handle while in the middle of puppeteering a robot and interacting with someone. Recall the example of the unwanted reflexive feet-look-at behavior while speaking with a person through the robot. One improvement we could make would be to create a suggestion system that would suggest reflexive behaviors to the puppeteer if the puppeteer is controlling the robot. If the puppeteer was away from the computer (which could be detected by monitoring usage on the interface) then these reflexive behaviors would be done autonomously, but if the puppeteer was using the

interface, then the system would make suggestions about behaviors to execute. The puppeteer then could confirm the suggestion at the click of a button. Another way to describe this interface would be that the autonomous behaviors are sensitive to the context of whether the puppeteer is using the interface or not.

Many times during demos, onlookers would try to get the robot's attention by waving their hands in front of the robot's face. In order for the robot to recognize this sort of attention-grabbing gesture, motion detection algorithms could be included from the OpenCV library. When the robot detects highly variable motion in a local area, the robot could move its gaze to the centriod of the motion (or suggest this to the puppeteer if the suggestion interface described above is being used).

Another feature that would be invaluable when talking to multiple people is detect and recognize faces. At the click of a button, the puppeteer could select to track a particular person interacting with the robot. This way, the robot would ignore any social cues from other users in the remote environment, and only focus on that one person. Labels identifing users could float on the stale panorama and would follow the image of the person's face as they moved about. A multi-modal use of video and sound could even be used to identify the speaker and allow the robot to always autonomously track the current speaker as is done in this system [21].

Near-Future Applications

The Huggable project's scope extends beyond just the social communication application. Because the Huggable project is designed to be a general robotics platform, there are very few restrictions that prevent it from being tailored to other applications.

Health Care

The Huggable robot could be used in the hospital room setting where it could become an extension of the nursing staff. Given its small teddy bear form factor, it can serve to calm and soothe child patients during their hospital stay. With the skin sensors that the robot offers, a patient who might not be able to vocalize where they feel pain might squeeze the robot in a location that corresponds to where they feel the pain. The touch classification algorithms currently in development could detect when this event occurs and the robot could subsequently alert the nursing staff. Using its puppeteering capabilities, a nurse could potentially monitor and several patients at once, allowing him or her to take control of any one of the robots to interact with a patient.

Huggable robots could be taken home from the hospitals to record a patients recovery from a major operation or treament. These robot's could potentially remind the patient to take their pills or record their blood pressure. In treaments like chemotherapy, it is important for patients to record how they are feeling everyday. The Huggable robot could help remind the patient to record these data, and the robot could even handle sending the data to the doctor immediately. Thus, providing the hospital staff with more information in order to better the health care they give to their patients.

Education

Reading with Rover is a program based in the Northwest that helps children with a difficulty in reading. The way it works is that a child reads stories to a trained dog who sits and listens. Its success comes from the idea that the children have an easier time reading to a dog as opposed to their peers. The child feels that the dog does not judge his her reading ability [20]. I believe the Huggable robot can achieve this same effect with some minimal autonomous behaviors akin to a pet who sits by the child side and makes some idle gestures. In fact, the robot's behavior could be enhanced by responding the child's tone of voice as a measure of the emotion in the story.

The robot could also be used as a conduit for education material. For example, there are geographic regions where it is difficult for teachers to travel from one region to another and hence education can not be properly given to children in these hard to reach regions. Such a region is the Highlands and Islands of Scotland [32]. The community there is interested in teaching the ancient language of Gaelic to its younger generations. One of the problems with this endeavor is that the experts in Gaelic are few and usually live far away from these remote communities. Highlands and Islands Enterprise (HIE), a company invested in the welfare and social well-being of Scotland, is interested in helping bring Gaelic to more of these remote communities. HIE has been seeking to invest in new technologies to help solve this problem. Fortunately, the remote islands and highlands of this region are connected via broadband internet. This internet connection infrastructure could be leveraged by the Huggable platform to allow teachers to log into these robots and deliver educational material.

Children with autism have difficulty adjusting to new teachers when the graduate from one grade to the next. The Huggable robot could be used to deliver educational material to these children by allowing teachers to puppeteer the robot. Furthermore, since the child interacts with only the robot, the constant and familiar form will help the child adjust to changes in teachers.

Entertainment

Expert puppeteer's could make a theme park come to life by controlling Huggable robots that greet, give information, and entertain visitors. These robots could come in a variety of form factors to represent the different characters of the theme park, and yet they would utilize the same puppeteering technologies developed for this thesis. Furthermore, using the technologies that allow the puppeteer to understand their remote environment and understand how they are being interacted with, visitors could even pick up these robots and take them along as a personal theme park tour guide. Puppeteers, could give advice on which rides to go on, or what restaurants have the food the visitors crave. These robots could even listen to all the places the visitors want to go to for the day and plan a shortest

path around the park that would visit each place. And with sophisticated enough semi-autonomous behaviors, a very skilled puppeteer could even control multiple robots at once. The Huggable robots could provide the park with a rich and entertaining experience for their visitors and even the puppeteers.

Robots have long been used in the film industry in movies such as Jurassic Park and Star Wars. Sometimes these robots can take a team of people to control all of the robot's degrees of freedom during filming. The puppeteering technologies designed here, such as the stale panorama and technologies designed by other members of my research group, such as the embodied puppeteering interface could reduce the size of the team to one.

Industrial Robots

The extensibility of the Huggable platform could be used to provide software for a variety of industrial robots. The diagnostic and monitoring pages could allow technicians to maintain their shop floor's robotic workers by providing with the information they need to diagnose problems quickly. Puppeteer's could use the interfaces of the Huggable platform to have the robot perform dangerous tasks needed for their job. The intuitive nature of the Huggable puppeetering technologies could reduce the amount of training these puppeteers would need. Developers could quickly incorporate new sensors on their robot by designing MSRS *services* for them and running them along side the rest of the Huggable platform.

Conclusion

This thesis presents a robotic framework that is designed to be extensible and robust. In addition, the framework was tailored to fit the requirements of a semi-autonomous robot-mediated social communication application. I have described the sorts of technologies that make the platform easy to maintain, extensible, and capable of integrating legacy technologies. I have also presented a series of technologies that make the remote social communication activity engaging for the user as well as for the puppeteer. Many of the

technologies described here aid the puppeteer in the complex activity of controlling a sophisticated robot and helping him or her to understand how the robot is interacted with. Some interesting engineering problems included designing a framework to work on a difficult platform such as the internet, designing an intuitive and novel puppeteering user interface that would not assume any skill of the puppeteer, and exploring the idea of semi-autonomy with a sociable robot.

Certainly, the performance of this system was not thoroughly tested except in an informal user trial. However, even an event such the making of a demonstration film produced insightful knowledge about which parts of the system worked and which ones needed improvement. In retrospect, it would have been smarter to have developed a defined set of metrics on which to evaluate the system. While there were requirements defined for the social communication application, there was no guidance on how to measure the degree of success in meeting those requirements. Furthermore, the degree of success of the platform was measured against how well it allowed for the development of the social communication application.

Nonetheless, the design and development of such a large and complex robotic platform provided me with countless hours of experience and stretched my knowledge to span over many different fields outside of computer science and engineering such as mechanical engineering, electrical engineering, and psychology. The multi-disciplinary nature of robotics allows me to work with constraints beyond the computer software and internalize the insight gained from such work to apply in future endeavors. The opportunity to design large systems such as the one created for this thesis is rare and I am grateful that I was given such an oppurtunity.

The Huggable project will be an ongoing endevour for many more years. I am happy that children who have had the chance to briefly interact with the Huggable robot have shown joy and enthusiasm for it. I believe that the project is a worthy one and necessitates the need for talented students to continue its success. The aims of this project are universally supported by communities and governments around the world, and I hope astute and

prudent investors will realize its impact and join in the support of this project for the years to come.

Acknowledgments

I thank the members of the Personal Robotics Group who have guided and inspired me in the field of robotics. I thank Allan Maymin for his extensive computer programming talent and knowledge. I thank Lily Liu for her wonderful teddy bear drawings that made the puppeteering interface the what it is. And finally, I thank my brother, Steve Toscano, for his insightful guidance in the field of Computer Science and his constant deep interest in my studies.

Bibliography

1. Apple, "Dynamic HTML and XML: The XMLHttpRequest Object", The Developer Connection, http://developer.apple.com/internet/webcontent/xmlhttpreq.html.
2. Blumberg, Bruce, et. al., "Integrated Learning for Interactive Synthetic Characters", ACM Transactions on Graphics, 2002.
3. Brave, S., Dahley, A., "inTouch: A Medium for Haptic Interpersonal Communication", http://tangible.media.mit.edu/papers/inTouch_CHI97.php, CHI 1997.
4. Breazeal, Cynthia L., *Designing Sociable Robots*, The MIT Press, 2002.
5. Breazeal, Cynthia L., et. al., "Using Perspective Taking to Learn From Ambiguous Demonstrations", Journal of Robotics and Autonomous Systems Special Issue on Robot Programming by Demonstration, 2006.
6. CereProc, http://www.cereproc.com/.
7. CuteCircuit, The Hug Shirt, http://www.cutecircuit.com/projects/wearables/thehugshirt/.
8. DiSalvo, C., et al., "The Hug: An Exploration of Robotic Form for Intimate Communication", in Ro-Man 2003.
9. Fischler, Martin A. and Bolles, Robert C. (June 1981), "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", http://portal.acm.org/citation.cfm?doid=358669.358692, 1981.
10. Gamma, Erich, et. al., *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley Professional, 1994.
11. Goza, S. M., et. al., "Telepresence control of the NASA/DARPA Robonaut on a Mobility Platform", *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, Vienna, Austria, ACM, 2004, pp. 623-629.
12. Hancher, M, "A Motor Control Framework for Many-Axis Interactive Robots", MIT Masters Thesis 2003.
13. Interbots. http://www.etc.cmu.edu/projects/ibi/.
14. Ishiguro, H., et. al., "Andriod as a Telecommunication Medium with a Human-Like Presence", HRI 2007, Arlington, Virginia, USA, 2007, pp. 193-200.
15. Kidd, C., "Sociable Robots: The Role of Presence and Task in Human-Robot Interaction", in MIT Media Lab Master's Thesis. 2003, MIT: Cambridge, MA. 2003.
16. Lee, Jun K. and Toscano, Robert L., "The Design of a Semi-Autonomous Robot Avatar for Family Communication and Education", Accepted in Ro-Man 2008.
17. Lowe, David G., "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, 60, 2 (2004), pp. 91-110. http://www.cs.ubc.ca/spider/lowe/pubs.html, 2004.
18. Nielsen, J., and Molich, R., Heuristic evaluation of user interfaces, *Proc. ACM CHI'90 Conf.*, 1990, Seattle, WA, 249-256.
19. OpenCV, OpenCV Library Wiki, http://opencvlibrary.sourceforge.net/
20. Reading with Rover, http://readingwithrover.org/
21. Siracusa, M., et. al., "A Multi-Modal Approach for Determining Speaker Location and Focus", ICMI 2003, http://groups.csail.mit.edu/vision/vip/papers/Siracusa_icmi2003.pdf, 2003.
22. Somby, Michael, "Comparison of Robotic Frameworks", http://www.linuxdevices.com/articles/AT5739475111.html, Aug. 17, 2007.
23. Sony, Sony Aibo, http://support.sony-europe.com/aibo/1_2_library.asp.
24. Steinmetz, Ralf, "Human Perception of Jitter and Media Synchronization", IEEE Journal on Selected Areas in Communication, Vol. 14, No. 1, 1996.

25. Stiehl, W. D., et. al., "Design of a Therapeutic Robotic Companion for Relational, Affective Touch", IEEE International Workshop on Robot and Human Interactive Communication, RO-MAN 2005, Nashville, TN, USA, 2005.
26. Stiehl, W. D., et. al., "A 'Sensitive Skin' for Robotic Companions Featuring Temperature, Force, and Electric Field Sensors", IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, Beijing, China, 2006.
27. Thrun, S., et, al., "Robust Monte Carlo Localization for Mobile Robots", Aritficial Intelligence (Journal), 2000.
28. Thrun, S., Robotic Mapping: A Survey, Exploring Artificial Intelligence in the New Millenium, 2002.
29. Wikipedia, "ACID", http://en.wikipedia.org/wiki/ACID.
30. Wikipedia, "Audio Frequency", http://en.wikipedia.org/wiki/Audio_frequency.
31. Wikipedia, "Fittz's Law", http://en.wikipedia.org/wiki/Fitts%27_law.
32. Wikipedia, "Scottish Highlands", http://en.wikipedia.org/wiki/Scottish_highlands.
33. Yoshino, K., "Disney Re-Animates Theme Park with No Human in Sight", *The Seattle Times,* Seattle, WA, 2007.