

April 2011

The Shadow Mirror

Anton Konstantinovich Zalutsky
Worcester Polytechnic Institute

Blake Zdralewicz Reeves
Worcester Polytechnic Institute

Warranty Walton
Worcester Polytechnic Institute

Xiaoli Ma
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/iqp-all>

Repository Citation

Zalutsky, A. K., Reeves, B. Z., Walton, W., & Ma, X. (2011). *The Shadow Mirror*. Retrieved from <https://digitalcommons.wpi.edu/iqp-all/2102>

This Unrestricted is brought to you for free and open access by the Interactive Qualifying Projects at Digital WPI. It has been accepted for inclusion in Interactive Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

THE SHADOW MIRROR

Interactive Qualifying Project Report completed in partial fulfillment
Of the Bachelor of Science degree at
Worcester Polytechnic Institute, Worcester, MA

Submitted to
Professor Rosenstock

By


Xiaoli Ma



Blake Reeves



Warranty Walton



Anton Zalutsky



April 22, 2011

Joshua Rosenstock

Abstract

The shadow mirror project is an interactive art installation which mimics user motions and displays them in the form of a puppet's shadow. In order to accomplish the shadow effect, a motion capture solution is needed alongside a puppet with a controllable motion system and lighting effects. This piece is open to a wide range of audiences of all ages and cultures, allowing us to achieve our goal of bringing forth a universal bond and introducing a new way of interaction through the blending of technology and art.

Authorship Page

As a whole, we worked together as a team. However, some responsibilities were divided according to each person's technical specialty. Xioali's specialty is in software development. She is mostly responsible for writing the programs in C++ and C#. Blake's specialty is in mechanical design. He is mostly responsible for the 3D modeling and mechanical drawings. Warranyu's specialty is in hardware implementation. He is mostly responsible for wiring and writing code for the microcontroller. Anton's specialty is in fabrication. He is mostly responsible for getting parts made and assembling them.

Acknowledgements

We would like to thank the Boston Museum of Science and iRobot for allowing us to attend the 2011 Robot Block Party and share our project with all those who attended. The experience was definitely worth-while. It allowed us to understand how people interact with our project and potential situations that can limit its performance. The event was very exciting for all of us and we all appreciate the opportunity to participate.

We would also like to thank our advisor, Prof. Rosenstock, for helping us stay on track and focused, not to mention notifying us of the great opportunity to show our work at the Robot Block Party.

Table of Contents

Abstract.....	2
Authorship Page.....	3
Acknowledgements.....	4
List of Figures	6
Executive Summary.....	8
i. Introduction	8
ii. Methodology.....	9
a. System Overview.....	9
b. Sensor Input	10
c. Microcontroller	11
d. Puppet.....	12
iii. Results.....	12
iv. Conclusions and Recommendations	13
1. Introduction	14
2. Literature Review/Background	16
2.1 Cultural History	16
2.2 Robotic Puppets	19
2.3 Interactive Art	22
3. Methodology.....	27
3.1 Physical Element	27
3.1.1 Puppet Chassis	27
3.1.2 Lighting Setup.....	29
3.2 Computer Vision System.....	30
3.2.1 Overview	30
3.2.2 Motion Capture.....	32
3.2.3 Data Communication	33
3.2.4 Data processing.....	33
3.2.5 Implementation and approaches.....	34
3.2.6 Limits and pitfalls	34
3.2.7 Control program class diagram	35
3.3 Control Systems	36

3.3.1	Scaling and Limiting the Position Coordinates.....	36
3.3.2	Translating Coordinates into Servo Angles	38
3.3.3	PC – Microcontroller Communication	39
3.3.4	Communication Diagrams.....	40
3.3.5	Translating Angle Data to PWM.....	40
3.3.6	Output Circuitry	41
4.	Results and Discussion / Analysis.....	42
5.	Conclusions	42
6.	Recommendations	43
	Appendices.....	45
	Appendix A: Part list and cost	45
	Appendix B: Pictures of Parts.....	46
	Appendix C: Control System Code in C#	48
	List of files	48
	form1.cs	48
	Appendix D: STM32 Code	56
	Main.c	56
	Global.h.....	60
	Global.c	61
	stm32f10x_it.c	68
	References	71

List of Figures

Figure 1	The Shadow Puppet	8
Figure 2	High Level Overview.....	9
Figure 3	Components of Shadow Puppet	10
Figure 4	OpenNI allows for tracking users and understanding limb positions.....	11
Figure 5	Minuet in pink, a concert selection of Lords "International "	16
Figure 6	Puppet Involving in Education ^[2]	17
Figure 7	Chinese Piyongxi.....	17
Figure 8	The Robot in "Skeletal Reflections"	18

Figure 9 A scene from "The Fishboy's Dream"	21
Figure 10 Egerstedt's Puppet	21
Figure 11 Keng's system	22
Figure 12 Ghost Pole Propagator displayed on wall at Belsay Hall Castle, Newcastle, England ^[9]	23
Figure 13 Shadows by Simon Briggs, interacting with a person ^[10]	24
Figure 14 Wooden Mirror by Daniel Rozin, mimicking person's face ^[11]	24
Figure 15 Solidworks® Model Front View	27
Figure 16 Intersection Detail.....	28
Figure 17 Side Detail, Hole/Wing Feature	28
Figure 18 Full Shadow Puppet Display	29
Figure 19 Program Layout.....	31
Figure 20 Program Flow Diagram	32
Figure 21 Control Program Class Diagram	35
Figure 22 Puppet's Pulleys' labeled	37
Figure 23 Servo Ranges	38
Figure 24 Communication Line	40
Figure 25 Servo Driver Circuit	41
Figure 26 A pulley integrated onto a servo for better movements of the string attaché	46
Figure 27 Back view of the top servos mounted on the puppet frame.....	46
Figure 28 A ball representing a hand of the puppet, attached to a spring and 2 servo motors	47
Figure 29 An old Bell & Howell Project 'N' View 500 projector.....	47

Executive Summary



Figure 1 The Shadow Puppet

i. Introduction

The Shadow Mirror was an IQP done by four students from Worcester Polytechnic Institute. The intended goal of this project was to create a unique art piece by utilizing the individual talents of each team member. The concept behind this project was to create the illusion of a shadow puppet that is controlled by a given user through the mimicking of the motions and actions of the user. The Shadow Mirror as built consists of an Xbox Kinect¹, a computer, a microcontroller, and a set of servos.

The concept of puppetry and public art goes back hundreds of centuries. Blending the past with the technology of the future, we were able to show our generation's implementation of something that hundreds of years ago would have been performed out on the streets. Art and technology have always been seen on different ends of the spectrum, but this project has shown that art and technology can indeed work together. The synthesis of an abstract and artistic concept with a dynamic and engineered product allows the Shadow Mirror to transform the common image of soulless robotics into a new form of technology that can unravel emotion from people of all ages and mindsets.

¹ Kinect has a set of 3 cameras which is able to track a person in 6 dimensions.
[Accessed 04/28/2011] <<http://www.xbox.com/en-US/kinect>>

ii. Methodology

a. System Overview

The Shadow Mirror is a closed looped system that allows continuous use from different users. At any given point a user may come into the loop of the system and will

be calibrated. At this point the system will begin its

interaction with the user and attempt to mimic his/her actions. The hierarchical system is branched into three sections:

1. Sensor input:

A stick figure representation of the user gives relative body positions to the computer after a configuration of the user.

2. Computation:

The computer uses positions of the hands and feet and converts them into angle positions of servos located on the frame. The values are then sent to an off board microprocessor via serial communication.

3. Actuation:

As the microprocessor receives joint positions, it converts them into pulse width modulation signals which are sent out to the servos and actuated.

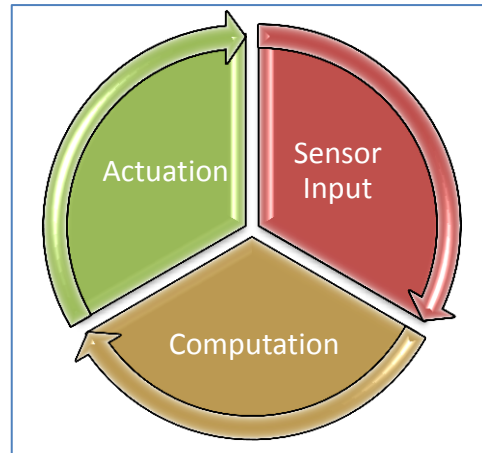


Figure 2 High Level Overview

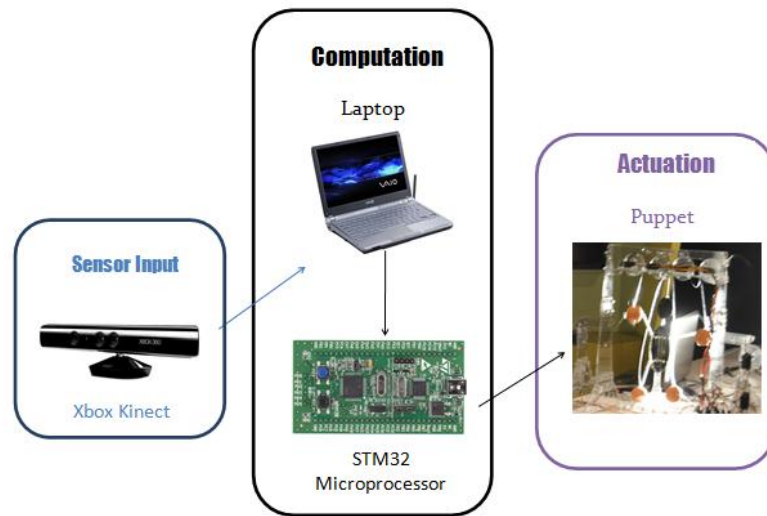


Figure 3 Components of Shadow Puppet

b. Sensor Input

The Xbox Kinect is a webcam-styled game controller which is used on the Xbox 360; however its capabilities are far more versatile. Using its stereovision, infrared sensors, and a software tool called OpenNI², a user's body position and relative limb positions are extrapolated in a stick form.

The sensor system described can track multiple users; however, our application allows only one user to be tracked at a time. In order for a user to be tracked, he/she must first go into a certain pose and wait for the OpenNI software to calibrate. Figure 3 shows an image of multiple users being recognized, but only user 1 (labeled 1-Tracking) is tracked by Kinect by certain pose.

² OpenNI is Wrapper written in C++ as the interface of Kinect.
 [Accessed 04/28/2011] <<http://www.openni.org/>>



Figure 4 OpenNI allows for tracking users and understanding limb positions

The main software is built around the OpenNI software, which is the interface for the Kinect. It is a combination of two programs. The first is a C++ program which takes the data from OpenNI through a socket and finds the limb positions relative to the torso. The second program, written in C#, translates this data into hand and foot positions for the puppet and calculates the necessary servo angles. This information is then passed to the microcontroller via a serial connection.

c. Microcontroller

The microcontroller is used to translate the data from the laptop to PWM signals that control each of the eight servos. The microcontroller used is the STM32VL-Discovery development board³. Data from the computer giving the positions for each servo is passed to the microcontroller via a serial communication line. This data is processed and converted into PWM signals which are sent to eight separate servos located on the puppet's frame.

³ The product information can be found from the site below.
[Accessed 04/28/2011] < <http://www.st.com/internet/evalboard/product/250863.jsp> >

d. Puppet

The puppet consists of a puppet, frame, and eight servo motors. The frame and servos act as the actual puppeteer. The eight servos control the motions of four balls which represent hands and feet of the puppet while the head and body of the puppet remain stationary. The puppet frame and puppet were designed out of acrylic because it was financially feasible and readily available. To make relocating the project simple, the frame can be split the off into two sections for ease of transportation. For stability, a large base was made to prevent unwanted forces and to keep it stable. This design was manufactured using a laser cutter.

iii. Results

As a user steps out in front of the Shadow Mirror, he or she must move into the required pose for calibration. After a few seconds, the shadow upon the screen comes alive. The lifting of the arm causes the shadow of a circle representing the hand to rise up as well. There is some delay and the speed is often slower than the user, but it follows the user's hand nonetheless. The total range of movement for each limb is limited to certain areas. For example, the Shadow Mirror cannot cross its arm over to the other side of the body. But even with its limitations the Shadow Mirror is still an enjoyable display to play around with.

From the experience at the Boston Museum of Science Robot Block Party, we find that people were drawn towards the Shadow Puppet, especially kids. As a small crowd looked on while they were waiting their turn to control the puppet, one could see the awe and sense of wonder at the moving shadow. A once dying art form had been renewed and merged with modern day technology. Many young kids who have never seen a real shadow puppet got to experience shadow puppetry in a completely new light. The Shadow Mirror also created a new way of interacting with and controlling a

shadow puppet. Whereas historically puppets were controlled by hand movements, the Shadow Mirror allows the user to control it with their entire body. This also allows for the user to express his or her individual creativity through the controlling of the puppet. The shared experience between the audience members promoted conversation between complete strangers and created sense of community.

iv. Conclusions and Recommendations

Overall, the Shadow Mirror accomplished its goals successfully. However, even though the results were satisfactory, there are improvements that could be made to enhance the system. One possible improvement would be to add a function where the puppet would perform its own actions to give life and personality to itself. Other improvements include more precise tracking, a puppet frame that allows for three dimensional movements with wider range, and a faster response time for the puppet. We believe that with these improvements, the Shadow Mirror will draw in even more people and perhaps even inspire others to explore the possibilities of the technology and make their own.

1. Introduction

Throughout history, public art has played a large role in the lives of people from all backgrounds. From the ancient Roman sculptures that decorated palaces and temples to the floating lanterns flown by thousands of Chinese as part of cultural festivals, the display of public art has been something that has enriched the lives of people for millennia. One of the earliest forms of public art that had a significant influence on the lives of people is puppetry. Puppetry has been around for hundreds of years and has served the role of being a medium for the art of storytelling. One of the reasons why puppetry was especially appealing to people compared to other forms of public art is its deep involvement with its audience. Whereas most traditional forms of art such as paintings, sculptures, and music only involved the passive observation of the artwork to convey its commemorative or decorative meaning, puppetry was the beginning of a new interpretation of art. Along with theatre, puppetry reacts and adapts to the mood and interests of the audience, allowing the audience to influence the performance of the artwork. From that point on, a new form of art was born. This form of art is called interactive art, as it emphasizes the involvement of the participants in conveying its artistic message.

Puppets interact with both puppeteer and audience by means of controlling and feedback. By giving life to puppets through manipulating various motions, puppeteers express their inner feelings and present their form of art in a unique way. Audiences could communicate with the puppeteers through puppets, disregarding the estrangement between people, introducing a new medium of communication.

Although puppetry and theatre has existed for thousands of years, the beginnings of a new form of interactive art began with the synthesis of traditional interactive art and robotic art. In the 1960's with the development of electronic control systems, the first generation of robotic art was born. As robots came into use in industrial production systems, artists took notice of these new inventions and explored the possibility of expanding the horizon of art by using these new machines as a medium for their work. ^[1] Paik and Abe's Koche's number 456, a "20-channel remote-controlled anthropomorphic

robot”, is a good example of this early form of robotic art. As technology became more and more sophisticated, the possibilities of more intricate control of machinery became possible. Within the last few decades, technology reached a certain point where it was possible for machines to mimic certain human movements. This has inspired artists to begin to rethink and recreate older art forms like puppetry, fusing together modern robotic art with a traditional interactive art. Drawing from that inspiration, we have decided to take the robotic puppetry idea further and create a whole new interpretation for this new art form, the Shadow Mirror.

The Shadow Mirror is a new kind of robot that draws from the ancient traditions of shadow puppetry while combining modern interactive technology.. It consists of a translucent wall on which the shadow is projected. The puppet itself is a small, electronically controlled two dimensional figure with a high intensity Bell & Howell projector behind it. Like the careful observation of the puppeteer, an Xbox Kinect is mounted to monitor the actions of the audience and mimic them in ways that give life to the shadow puppet. In this way, the audience can interact with and become more connected to the art piece. The puppet’s goals of inspiring and expressing emotion within the audience are enhanced with the life size shadow the puppet casts. This allows the user to feel and believe that the puppet is more human than it actually is, while still featuring a friendly cartoon-like shape. The mystique of the shadow only deepens the connection as the audience questions how the shadow is actually cast. As the audience peers around the screen to see plastic and electronic parts connected through a computer, they realize that technology acts as a medium for which humans can see the robot as a non-threatening and comforting human-like entity.

The shadow mirror project will change the perception of art and technology by showing that technology doesn’t necessarily have to be for utilitarian purposes. Rather, it can be used as an artistic medium through which people can express themselves. Being an easily movable display, the shadow mirror will provide shared experience to a wide range of people by being on display in multiple places.

Since the shadow mirror can interact with more than just one person, it can bring individual audience members together by interacting with all of them at the same time. All in all, the shadow mirror will provide an uplifting, creative, and fun atmosphere for people from all demographic areas.

By creating the shadow mirror, we want to achieve the goal of embracing shared experience between people, blurring the division of technology and art, promoting friendship through collaboration, and introducing an innovative way of interacting. As an opportunity to practice our expertise, the project itself provides us a platform of creating and learning as well. As a free-choice and open-ended project, the Shadow Mirror drives us to always look for the possibility of discovering more. From the aspect of working with peers from different fields, the project helps us better understand our professional and ethical responsibilities, and promote better communication among colleagues.

2. Literature Review/Background

2.1 Cultural History

Puppetry has been an ancient form of art in many cultures for more than 2000 years. Although its origins are unknown, puppetry has been widely used in different purposes. Many countries have



Figure 5 Minuet in pink, a concert selection of Lords "International "

implementations of shadow puppets, like Indonesia, India, Malaysia, Thailand, France, Turkey, etc. A broad classification of puppets would include hand puppet, marionette, rod puppet and shadow puppets.

The survival of this art form is due to man's fascination with the inanimate object animated in a dramatic manner (Figure 6).^[2] The



Figure 6 Puppet Involving in Education ^[2]

hand puppet is simply the shadow of a performer's hands projected to a screen, which figures are made by utilizing different distances of the light source. A marionette is the wooden puppet, controlled by

puppeteers with strings. The puppeteer gives voice to the marionette through performing various movements and express feelings to audience. A famous example of marionette is Pinocchio. The rod puppet is similar to marionettes, but controlled with rods from either below or above. Shadow puppets are usually flat cut-out figures manipulated by rods behind the screen.

Puppets have been employed in all purposes including entertainment, education, advertisement, religion, etc. As an intuitive and vivid educational tool, puppets always interact with children in different ways. (Figure 7) In a similar manner, puppets also have been involved in religious teaching, propaganda and other approaches.

Being a business, one story of its rise and fall could be told from the point of view of puppeteers struggling



Figure 7 Chinese Piyongxi

to make a living through shifting patterns of wage labor and cost of living and the growing entertainment industry.^[3] Puppet plays were performed on the street amongst poor people while the dominating class disregards puppetry in the early ages. Once as a popular entertainment, puppet shows

were held at home amongst working-class to complement low wages. Family members usually assisted. This form of home-based puppetry represented an early commercialized leisure time activity for the working class.^[4] The performing of puppets is usually accompanied with music, scene setting, story and sound. Long since, shadow puppet shows are popular entertainment for families, spreading to all classes, ages and cultures. Theaters are frequently visited for numerous shows by families, which has great influence on public social entertainment. An example is Chinese Piyongxi, which is a popular traditional shadow puppet show that plays in theater, home or street. The figures are cut-out from leather with detailed motions and facial expressions. Puppets are controlled by puppeteer from behind a translucent screen with strings. To carry out the precision of motions and expression, the controlling strings could have as many as more than 20. Color can be introduced into the cut-out shapes to provide



Figure 8 The Robot in "Skeletal Reflections"

a different dimension and different effects can be achieved by moving the puppet or light source out of focus. Characters may speak or sing through the performance, expressing variable feelings and characteristics. A story could be as long as hundred years, and each puppeteer could control one or more figures. With the puppeteer's superb skills, the puppet can precisely mimic human motions. Puppetry may be accessed and enjoyed by people regardless of classes, ages and cultures (Figure 8).

The performance of a shadow play could also be in various forms, such as a silent movie, characters with performers speaking or singing, or with music and sound. One form of shadow puppetry, hand shadow puppet play, needs only a light source and a display screen. The performer mimics animals, people and all kinds of image with their hands. With the light

source in front, the shadow projected on the screen could be controlled, and taking advantage of different distances, various effects could be achieved.

Regardless of different genres of puppetry, it is widely enjoyed by people with various cultural backgrounds over human history. Diversified usages and purposes are achieved by all approaches. Puppetry emphasizes the sense of art by creating dramas with puppets, with the puppet itself expressing the puppeteers' artistry.

2.2 Robotic Puppets

Ask someone to describe what a puppet is and most likely they will tell you that it is some kind of figure that is controlled by the hands of a puppeteer. For more than a millennium this has been the definition of puppetry. As technology has progressed, more and more complex mechanical methods are employed in controlling a puppet; but even with the most sophisticated mechanical constructs, puppets remain, for the most part, limited to interacting whilst there is a puppeteer controlling it. However, with the invention of the computer and powerful microcontrollers, the puppet is now capable of being controlled by a different kind of puppeteer -- computer programs. Instead of muscles moving rods or strings that control the limbs of a puppet, the puppet is now controlled via electromechanical actuators. With the use of the computer, the puppeteer now has the option to have the puppet become autonomous in its performance. It is important, however, to distinguish that robotic puppetry is not merely a robot. Rather, it is the use of robots as the actors in a performance, just as marionettes or shadow puppets are the actors in a performance.

Though the idea of using robotics as a means of controlling puppets is in its infantile stages compared to the entire history of puppetry, it is catching on as a new way for artists who are interested in technology to combine the ancient art of puppetry with the recent developments in technology. Robotic puppetry is also catching the attention of more traditional technology developers and academics that are looking to apply their understanding in a creative manner. Of the artist camp, the

works of both Chico MacMurtrie and Survival Research Labs stands out as ones that creatively employs this concept of robotic puppetry. Of the technological developers' camp, Magnus Egerstedt's study of the control of autonomous puppets and Tay Boon Keng, Raymond, and Stefan Künzler's "Robotic Marionette Systems" provide the technical detail of robotic puppetry systems.

Chico MacMurtrie is a well-known artist who specializes in robotics as a medium for his artistic expression. Since he began creating his art in 1989, he has "exhibited in more than 20 countries worldwide, receiving support from more than 15 national, local, and international granting agencies, and 30 corporate sponsors".^[5] Skeletal Reflections, one of MacMurtrie's earlier works, is a complete robotic skeleton that responds to human interaction. Using a motion capture system, the art piece analyzes the viewer's body position, compares it to a database of twenty poses from classical art pieces, and goes into the pose which is most similar to the viewer's body position. To translate MacMurtire's work back into traditional puppetry, Skeletal Reflections is like a performance where a puppeteer who, upon looking at the viewer, recognizes the body position of the viewer and manipulates his puppet into a well-known classical pose. Though it may seem an easy task for a human puppeteer, it is an incredibly complex task for an automated computer.



Figure 9 A scene from "The Fishboy's Dream"

individual robotic “puppets”. In a particular performance called “The Fishboy’s Dream” in January of 2006, remote controlled robots called the “Sneaky Soldiers” were employed in a war scene surrounded by fire and explosions. During this performance, the clear raw violence of human nature is shown, surprisingly, with no humans in the scene. Each of the eight robots was drawn into a chaotic battle and as a result, suffered distinct “wounds”. The work also featured other robotic pieces involved in the story. ^[6]

Of the scientific camp, Magnus Egerstedt’s papers “Optimization of Multi-Agent Motion Programs with Applications to Robotic Marionettes” and “Optimal Timing Control of Interconnected, Switched Systems with Applications to Robotic Marionettes” provides insights into some of the reasons why scientists and technologists pursue the goal of controlling something as simple as a puppet. It turns out that, beyond the superficial



Figure 10 Egerstedt's Puppet

glance, controlling the movement of a puppet with a high level of abstraction is a very complex task. Each high level description of the puppet’s action must be broken down into small steps with information on the “characteristics, duration, and intensity”. ^[7] This requires heavy computation under the study of control systems and multiple layers of software abstraction. Though the main purpose of the studies is to analyze the methodology of implementing control systems, the use of robotic puppets

in this area is still a solid example of how the idea of controlling puppets with robotic systems is gaining popularity.

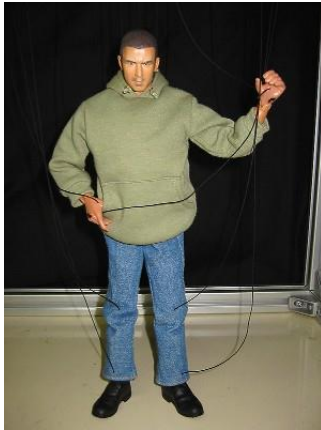


Figure 11 Keng's system

Yet another example of scientific research on robotic puppetry is Tay Boon Keng, Raymond, and Stefan Künzler's "Robotic Marionette Systems". The main objective of the study was to create a complete robotic marionette control system. The intent of the study was not so much the exploration of control systems for its own sake, but more for the actual implementation of robotic puppetry systems. According to

their research objectives, the "advantages of the robotized system are accuracy, repeatability and new variety of the marionette movements compared with manual operated marionettes" [sic].^[8] They also included that these systems will become a new and relevant art form since people will derive different meanings from a robotic puppet performance rather than a traditional one.

2.3 Interactive Art

Interactive media has been a revolutionary new interpretation of art in recent years. Its ability to morph and change based on given inputs allows for unique artistic experiences unlike those of normal stationary art pieces. This unique characteristic creates a new dimension to the piece which, in many cases, involves its surroundings and creates a much stronger bond with its viewers by making them or their environment a key part of the artwork. With this new interactive dimension introduced, a whole new outlet has been opened up to create unique forms of group and individual artistic expressions which can be manipulated by passers-by.

There are numerous ways to incorporate or interpret the theme of interactive art. The kind of interaction the viewers and environment have with the art is completely up to the artists themselves.

This means pieces can range from kinematic movements on the part of the users to sounds emitted from the environment to shadows cast by people walking past. The possibilities are endless. All over the world artists are utilizing various technologies that are available to them to create these unique outlooks and experiences for art-appreciators. Some examples include exhibits that capture and record user actions and interpret them into an alternate visual form to be reproduced in a unique perspective such as stick figures or wooden pieces. Other examples have created unique ways to utilize light, shadow, and in one case a person's own heartbeat to interact with an artificial intelligence.

In 2007 in Newcastle, England, an artist by the name of Golan Levin made an interactive piece inside the Belsay Hall Castle called Ghost Pole Propagator. In this piece, a motion detector capable of tracking the movements of each person in the room records the actions of the viewers and translates them into stick figures. In Figure 12, one can see how these stick figures and their motions are then projected on the dark dingy stone walls of the castle to represent a sort of modern-day cave painting that projects the recordings of events that took place at the site. ^[9]

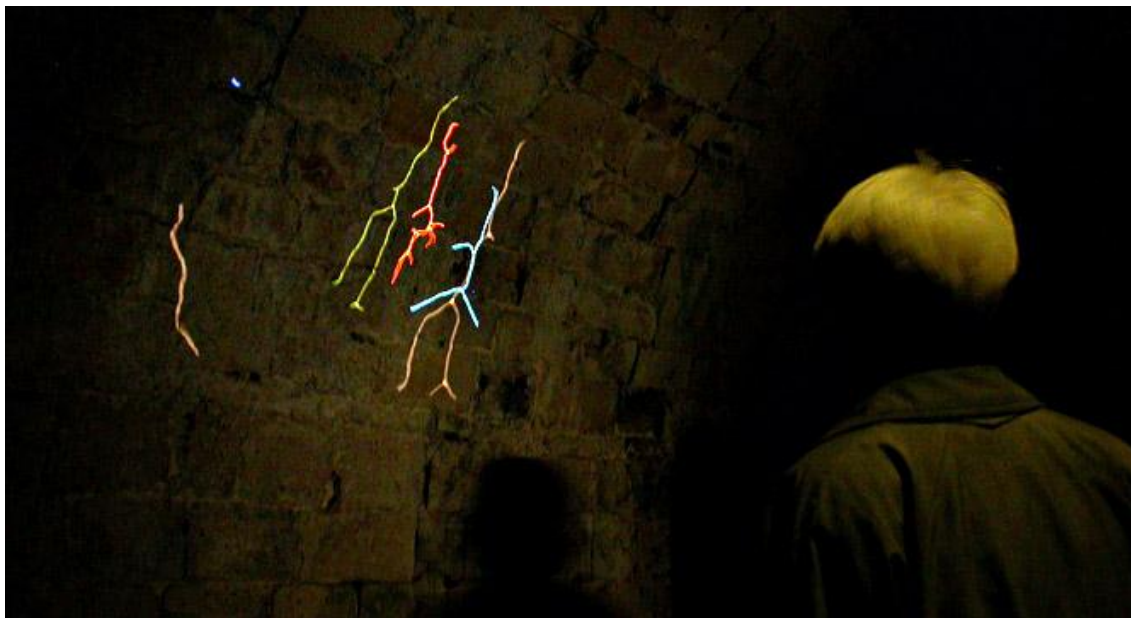


Figure 12 Ghost Pole Propagator displayed on wall at Belsay Hall Castle, Newcastle, England ^[9]

Another artist by the name of Simon Biggs simulated figures in his project “Shadows” which were projected on a wall as well, but instead of recording people’s actions he had computer-generated man



Figure 13 Shadows by Simon Biggs, interacting with a person ^[10]

and woman figures which reacted to their surroundings. These figures were depicted without clothes so one could easily tell the difference between the males and females. As viewers would pass by, their shadow would be projected onto the screen which would eventually collide with the placement of one of the virtual people. A photo of the general area is shown in Figure 13. ^[10]

Whenever touched by a real person’s shadow, the virtual figure would react to the collision and walk away. In addition, when one’s shadow collides with the figure, it would change its identity and more so, if the right side of the figure is where the collision occurs the figure will change its gender. This unique artistic expression can be deciphered into several possible interpretations which in itself creates a unique thinking experience for the users. ^[11]



Figure 14 Wooden Mirror by Daniel Rozin, mimicking person's face ^[11]

Another form of interactive art is a distinctive skew on normal video capture. Daniel Rozin created the “Wooden Mirror” in 1999. At first glance it just appears to be a bunch of small wooden squares in a picture frame hanging on a wall, but as one steps nearer each wood tile tilts in angle to effectively change color through the use of

shadows to mimic a TV screen, as shown in Figure 14.^[11]

This inspiring piece looks at a seemingly normal media and transforms it into a canvas for anyone to enjoy. The way Rozin creatively re-imagines the canvas is spectacular and captures the main goal of interactive art fully. He draws viewers in and makes them the art but does it in a way that not many people would think of, thus creating grounds for provoking new outlooks on certain media and the way it can be used.

One other interesting form of interactive art is the manipulation of virtual reality through the use of user inputs. A group of art students from Israel made a piece called “Heartbeats” in 2006 which consisted of four pillars surrounding a projected circle on the floor. In this circle there were four virtual people in the fetal position and when a user walked up to a pillar and placed his or her hands on it a heartbeat-monitoring device would sense their presence and bring the corresponding virtual person to life. Based on the user’s heartbeat the person would travel around the circle at different speeds. The different users could team up and try to get their virtual people to intersect one another’s path which would trigger an interesting dance sequence and restart the positions. This design is very interesting because it involves team work and gathering strangers to accomplish a goal, thus involving the viewers in the art thoroughly.

David Rockey is a well-known artist who investigated the four main iterations of interactive art that have been developed throughout its history. In his article *Transforming Mirrors* ^[12] he explains how interactive art is like looking into a mirror, but the artwork that comes out of it is a different form of refraction that the user can interpret. The four main methods he targeted were Navigation, Media, Mirror, and Automata. Navigation is a piece that requires the user to walk through a place and interact with its environment. The Media method allows the audience to express themselves through a new medium, whether it be digital, mechanical or something else. Mirror allows the user to see themselves or a silhouette within a digital media and interact in the virtual world. Finally, Automata are the

interaction between the users and an AI. Each of these styles all still create a new way to look at art and incorporate the user as a key part to a unique new medium. In our project we try to encompass all of these key methods laid out by Rockeby to create as many perspectives of interaction possible for the users.

It is clear to see that there are infinitely many ways to go about creating and interpreting interactive art and involving the user. It all depends on the audiences one wishes to target and how involved in the art they want them to be. Interactive art doesn't only have to be interaction with the art and users but can span to the surrounding environment including sounds, light, and anything else naturally occurring. The possibilities are endless.

3. Methodology

3.1 Physical Element

3.1.1 Puppet Chassis

The main structure was designed to serve as a chassis for both the puppet as well as mounting supports for the servo motors. Each limb of the puppet was intended to be controlled utilizing two servo motors and a spring to allow for more fluid movement. As a result, a frame-like structure was designed to surround the puppet. The entire design was made to be simple to assemble and strong, so each piece was made into a two dimensional profile that was easy to cut using a laser cutter.

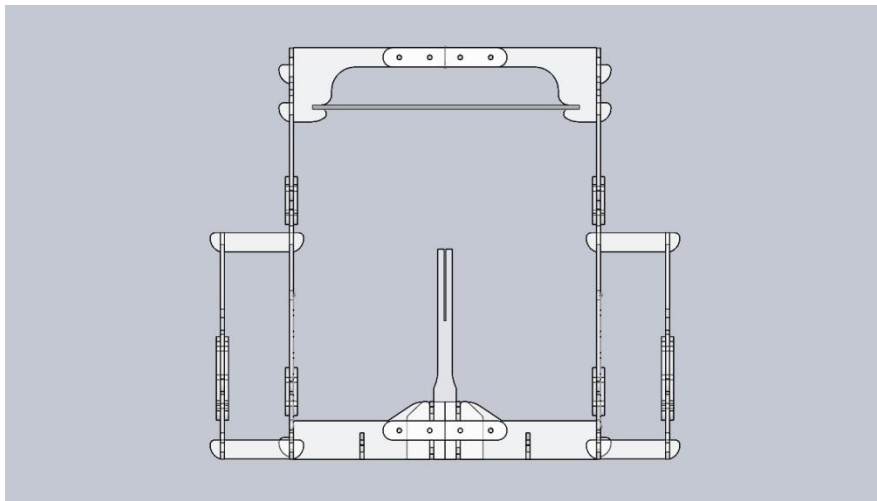


Figure 15 Solidworks® Model Front View

The frame features a top shelf which is used to support the four top servos. Side wings mount the remaining servos, while the center guide is the fixture for the puppet body (Figure 15). The entire frame structure is made out of clear acrylic plastic to ensure the frame would not interfere with the casting of the shadow and still have the strength to support the internal tensile forces involved. It is held together using a combination of bolted brackets, acrylic glue, and a unique puzzle-like assembly design which can be seen at the intersecting pieces in Figure 16.

To increase the precision of the shadow placement, a central guide was designed for the puppet. This guide prevents the movement of the puppet in the X and Y directions and minimizes its movement to less than an inch in the Z direction. This reduces stress in

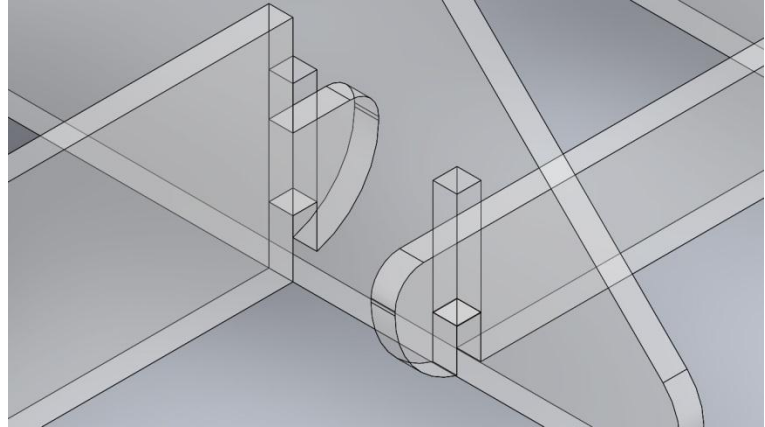


Figure 16 Intersection Detail

certain puppet positions. By making the puppet body out of acrylic and implementing a simple mounting bracket on the back of the puppet, it was able to travel on the guide like tracks.

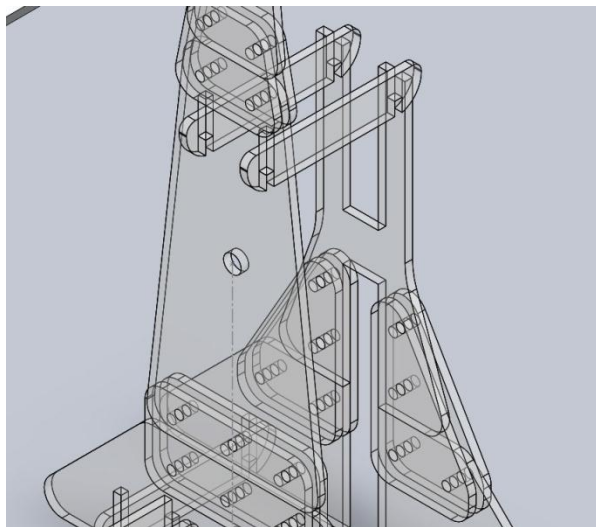


Figure 17 Side Detail, Hole/Wing Feature

Side wings were made to offset the side servos from the main chassis, which served several purposes. On each servo mounted is a pulley used to wind up or release a string controlling a limb. The side wings ensure that these pulleys would not interfere with the shadow. If the pulleys were mounted on the main inner frame, they would have a prominent impact on the overall display and the

mystique of the shadow would be minimized. Another important use for the wings was to allow the position of the string location to be limited to a certain location. In Figure 17, one can see that two holes are cut into the main inner frame on both sides; these holes are where the strings are fed through, connecting the limbs of the puppet to their corresponding servo motors. This is essential for the calculations of limb position because it limits the string position to one given point and eliminates the need to compensate for inaccuracies caused by the exact position the string may be on the pulley.

The top shelf was designed as a simple rectangular plate for a variety of reasons: first of which was to allow for easy replacement if a different material than acrylic would be necessary. The ability to remove the shelf and replace it later was a useful feature for assembly and transportation as well, as it allows for the installation of the servos separately. Finally, because it was essentially an empty shelf, the orientation and positioning of the servos was given much greater freedom, allowing modifications to be made to optimize how the system could work.

3.1.2 Lighting Setup

Several tests were done in order to find the optimum lighting source and orientation for the shadow display. The final setup consisted of a vintage Bell & Howell “Project ‘n View 500” projector as a light source mounted about four feet behind the puppet, and a screen⁴ with wax paper-like inserts mounted about 6 feet in front of the puppet (Figure 18).



Figure 18 Full Shadow Puppet Display

⁴ A Japanese Byōbu style folding screen with 3 panels.

Many light sources were tested in order to accomplish the best shadow and it was determined that a projector would be the best choice. Projectors have the characteristic of a clean and focused light, which displays a very crisp shadow outline. Other lamps such as flood lights and work lights were initially preferred, but the reflecting plates used in those styles tended to skew and distort the shadow too much to view correctly. Those lamps also tend to exude a large amount of heat, which could lead to several complications after long exposure. Using an old slide projector we were able to minimize cost, heat emanated, and keep a crisper shadow image

A Japanese Byobu style screen was chosen as the best option for a surface to project the shadow on. There was one available to the project and it had the feature of being able to stand on its own, not to mention being the correct size and visually attractive. The other option was to use a plain white shower curtain, but a stand or display would have needed to be constructed in order to use it properly. Due to time constraints, it was not feasible to create this stand. The best choice was to go with the screen, which proved to work very well according to several people who tested the project when it was on display at the Boston Museum of Science.

3.2 Computer Vision System

3.2.1 Overview

The Software system consists of two separate sets of software. The first is for the Kinect to capture and collect user position data. The second is the visual puppet and servo control system software. The visual puppet software is a window with six pictureBox⁵ objects representing hands, feet, head and body. The hands and feet pictureBox objects mimic the user by moving to new positions. A timer and a serial port are set up to receive data and to control communications. The visual puppet window is shown in the figure below (Figure 19).

⁵ A pictureBox is a standard picture container in C# form development.

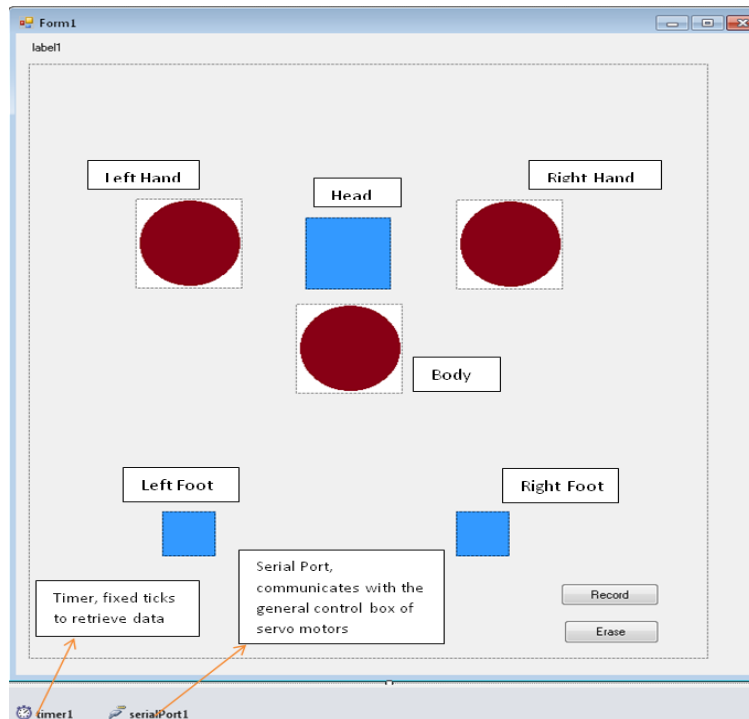


Figure 19 Program Layout

The Microsoft Kinect is the main component the vision system. The two software systems stream data in a client-server mode. The Kinect vision system, building on top of OpenNI, enables us to keep track of the positions of user’s hands and feet in real time and processes data to be sent to the control system. The control system receives data in periodic intervals controlled by a timer.. Data is received and processed with each tick of the timer. Processed data is then sent to the corresponding control block which updates the positions of the pictureBoxes. These positions will is then used to calculate the angles for the servos. Finally, the angle values are outputted via the serial port.

The program flows from the vision system software to the visual puppet software, which then parses raw data and navigate to distinct control blocks. Each control block further processes data, and flows through the common serial port to the microcontroller. A program flow diagram can be seen in the figure below (Figure 20).

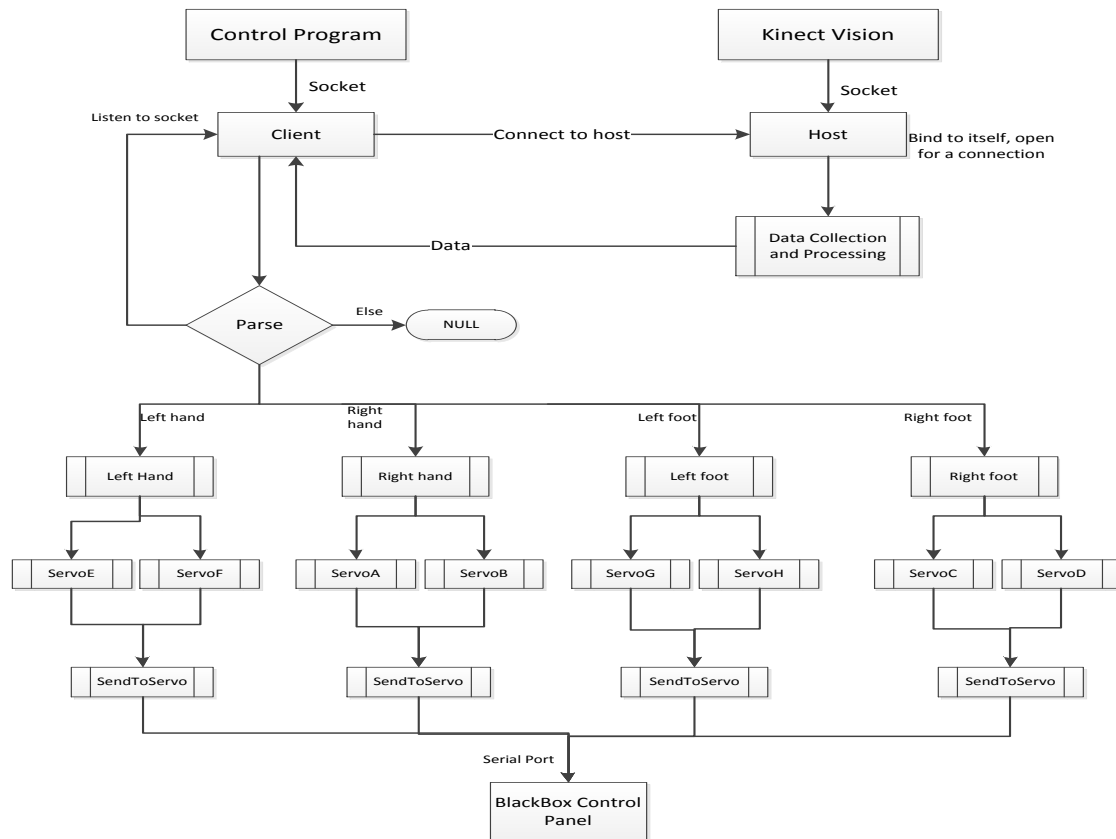


Figure 20 Program Flow Diagram

3.2.2 Motion Capture

Utilizing the Kinect’s ability of recognizing and tracking people, a user could be recognized in a certain pose in front of the camera. Twenty nodes representing different body parts can then be tracked, with each node outputting its coordinate values. A depth map of a user and the background will be drawn after the program is activated, and a skeleton in solid line is drawn after a user is tracked. As for our project, only the positions of hands, feet, and torso are needed. The difference between a hand or foot and the torso is calculated in three dimensions and tagged with a name representing the limb associated with that node. It is then formatted for communication to the control software system. Every position is updated each time the Kinect captures input.

As the program starts up, a TCP socket⁶ is created. It then waits for a client to connect to it, in this case the control software. The control software also initiates a socket to connect to the host on start up. After a connection is set up, the Kinect will be ready to capture. A user is detected by the program when they are in range and in the correct pose.

3.2.3 Data Communication

A TCP socket is used for secure data transfer and connection. There is exactly one host function at a time, and exactly one client connects to the host. A unicast is set up from the vision system, which is written in C++, to the control system written in C#. This guarantees that the data stream is isolated and will not be interrupted. The one way streaming of data from the vision program is sent in the form "x_dist,y_dist,z_dist, node_name," all in floating point. The same format of the receiving package is set from the control system. A parse mechanism is used to organize the data into a structure form.

3.2.4 Data processing

On receiving on the control system end, parsing would take the name, truncate data to integers, and drop the z dimension for our 2-D system. The program then directs to the corresponding control block for further functions. For each pictureBox representing either one of the hands or feet, a new position will be calculated from the given distance to the pictureBox representing the torso. Each of the control blocks in the program positions the corresponding pictureBox while sending the servo angle values that moves the actual puppet to the same position. Each servo motor could turn from 0° to 180° degrees, so a conversion from distance to angle is needed. The degrees to turn is calculated based on the information of distance to move, motor position , as well as dimensions of the pulley system assisting movements.

⁶ A TCP socket enables a secure communication between 2 programs by binding to a port

3.2.5 Implementation and approaches

C# is an object oriented and event triggered language. It also supports serial port communication in a simple and accessible way. The C# network library supports the communication with the Kinect program in a user friendly form. This type of development environment suits well to our visual puppet system. Furthermore, its serial port communication allows the integration of both software and hardware. Our first approach was actually to implement the program in Java with the Swing library. However, the serial port library for a 32 bit machine running Windows 7 is no longer supported. This blocked the integration of software and hardware systems. The second approach was done in C#, with mouse triggered movements, which was later replaced with input from the Kinect. A recording and replay system that allows users to record a sequence of movements and play it back afterwards was also developed and tested. A simple implementation of writing and reading file operations was approached and tested for the second build and also the final build.

3.2.6 Limits and pitfalls

The Kinect's capturing ability is limited by light intensity, noise, color pattern of the room, and clothing. Users may not be recognized if they are standing still when the program starts, since the Kinect tends to recognize human body movement better than stationary persons. After a user is recognized, sometimes the software may not be able to track all the nodes. Loose fitting clothing is a good example for this. In order to draw the skeleton by nodes any objects with bright colors were considered a potential user and asked for pose.

When a user is tracked, overlapping of body parts may cause the software to lose track of limbs, and the corresponding data will not be collected. Fast moving limbs may cause lag in figure and skeleton updates. Different poses such as turning to the side bending over, or twisting may cause distortion of the skeleton drawing. The according data collected would be corrupted and dropped.

The timer in the control system is set to run at 10 milliseconds, and it is updated 100 frames per second, whereas the Kinect captures at 33 frames per second. Assuming that the network is always secured and the TCP socket will never drop any data, the program is actually updating faster than the Kinect. Null data retrieval is expected but the performance is guaranteed to be fluid enough for human eyes. Although the PictureBoxes may move out of the working window, the range of puppet movement is fairly limited. A software method to account for the physical limitations is implemented and will be discussed later.

3.2.7 Control program class diagram

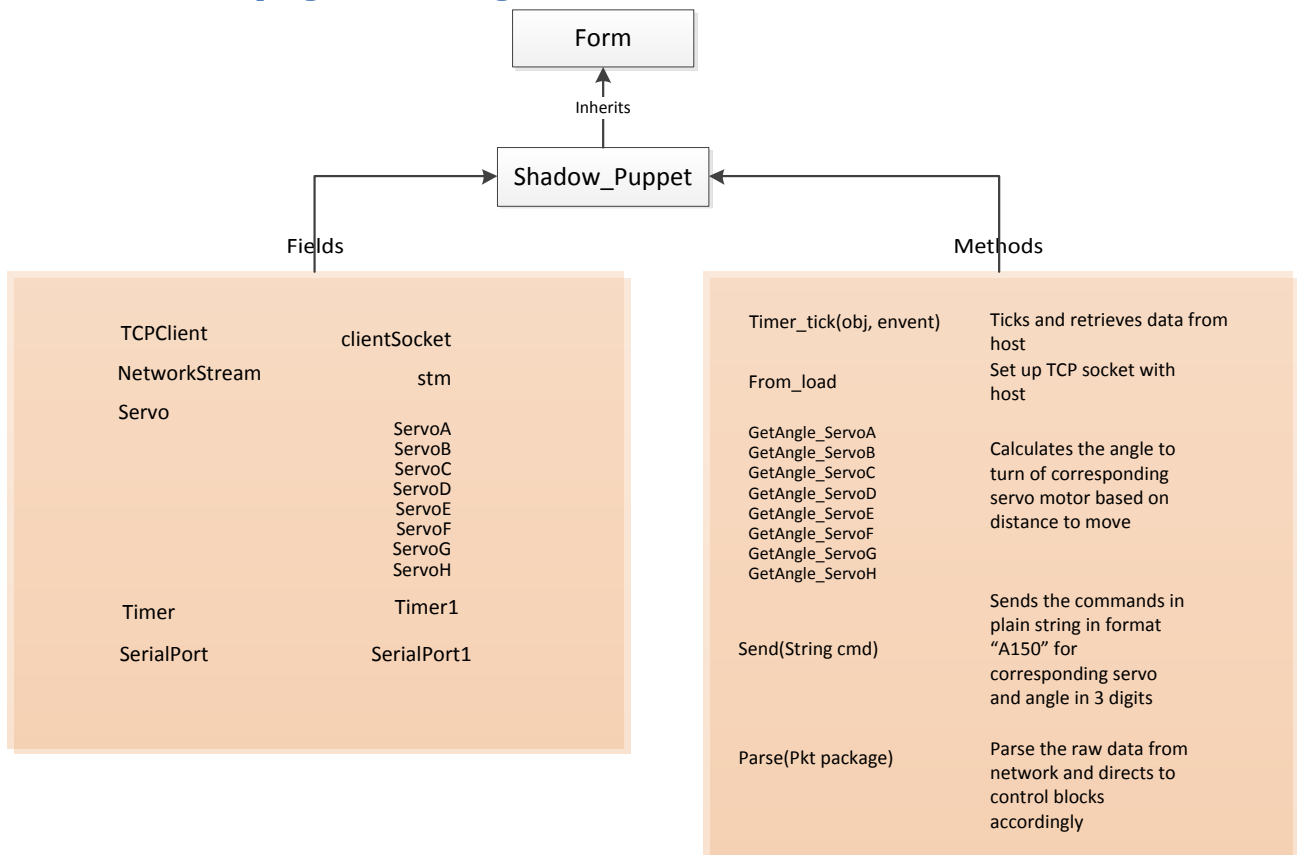


Figure 21 Control Program Class Diagram

3.3 Control Systems

3.3.1 Scaling and Limiting the Position Coordinates

During the transfer of data from OpenNI to the visual puppet control program, the data is passed through the vision system software that performs some preliminary calculations and regulates the data stream. While in this process, the vision system scales the X and Y coordinates such that the position coordinate of each limb responds better given the window dimensions of the visual puppet program. From experimentation, a scale value of 3.125 provides the best mapping from the Kinect world to the visual puppet world. Once this is done, the difference between the position of each limb and the torso is calculated. For each of the four limbs denoted rh, rf, lh, and lf, the difference between the limb and the torso is sent in this format: ΔX , ΔY , ΔZ , name. For example, “233, 15, 400, lh” would correspond to a positional difference from the left hand to the torso of 233 pixels in the X direction, 15 pixels in the Y direction, and 400 pixels in the Z direction.

The vision system then transfers this data string over a socket port. Data is taken in by the visual puppet program via a timer function. Every 10ms, the program reads the incoming data and parses them. The name of the limb and the X, Y, and Z values are stored in a structure variable (Z is not used). Once parsed, the timer function performs all calculations and outputs the data over the serial port. At this point, the program waits until the timer function is called again 10 ms after it was called previously.

Regardless of the input data from the Kinect™, there are physical boundaries where the system will not be able to match the physical piece of the puppet to the location of the limb that is received. In order to understand the limitations, the mapping of the physical system to the pixel representation must first be explained. The physical dimensions of the box, 16” x 17.5”, are translated into pixels by a scale factor of 40. This gives a working window of 640 x 700 pixels. The physical locations of the center

of the servo axis (with the exception of the servos on the sides, where the locations of the holes where the string comes through is used instead) are also located and stored as points on the working window.

Each of the servos is labeled as follows:

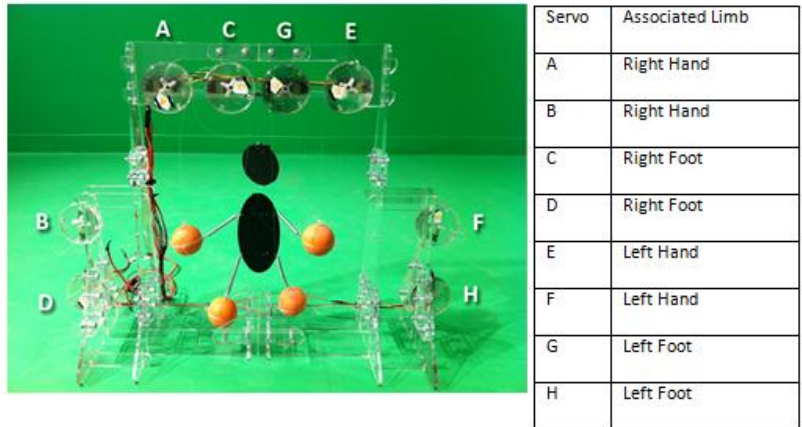


Figure 22 Puppet's Pulleys' labeled

The distance from the position of the limb (a point whose coordinates depend upon the input from the Kinect™) to the corresponding servos is calculated via the Pythagorean Theorem. The problem comes in when the total distance to be traveled is greater than the servos can physically allow, which is πR or 4.320" for a pulley radius of 1.375". The result is that the position of the limb must be limited within a certain range. Figure 23 shows the allowable range which the puppet limb can actually move.

Each servo has an offset of a certain distance to push the allowable area further inwards towards the center. However, this also means that the distance from the limb to the servo cannot be less than the offset distance. Only the overlapping area between both servos' ranges is a valid area to put the corresponding puppet

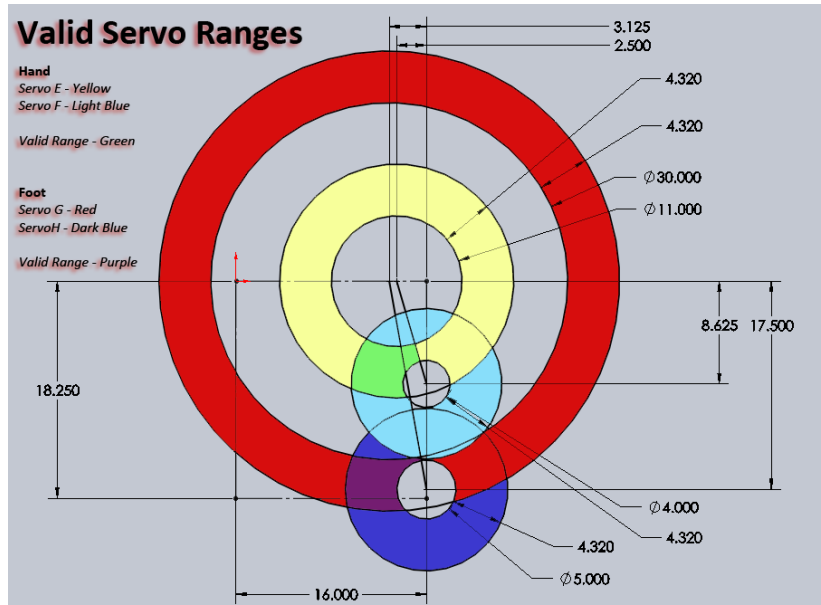


Figure 23 Servo Ranges

limb in. Any commands to do otherwise will cause a high current draw in the servo that is trying to pull a fully extended string. In order to avoid this, a simple if-then statement is used to determine the validity of the location of the limb. Basically, if the position requires either of the two associated servos to go beyond 180 degrees, the signal to the microcontroller is not sent.

Lastly there is the problem of gravity. Since the servos only work by winding and releasing the string, the area above the direct line between the servo pair is off limits; it would require a spring pulling the puppet limb from the opposite direction. This is easily corrected by another if-then statement where if the Y location of the limb would be less than the Y value of the line for the given limb's X value, the program ignores this input and waits for another input.

3.3.2 Translating Coordinates into Servo Angles

As mentioned previously, in order to get the appropriate servo angles for the puppet to be in the correct position, the distance of from the virtual limb to the associated servo pairs must first be calculated. This calculation must also subtract the offset string length used to put the puppet limb in a

desired position. The result is the actual length of string that the servo's pulley must release. The calculation follows this formula:

$$LengthToServo_x = \sqrt{(X_{Servo_x Location} - X_{Limb Location})^2 + (Y_{Servo_x Location} - Y_{Limb Location})^2} - Offset_{Servo_x}$$

Once the length required for each of the two servos is calculated, the angle can be easily obtained using the following equation:

$$Servo_x Angle = \left(\frac{LengthToServo_x}{R_{Pulley} \times \pi} \right) \times 180$$

Because of the different physical orientation, servos B, E, F, and H must output the calculated angle's supplement (180 – angle).

3.3.3 PC – Microcontroller Communication

When the calculations for the two servo angles associated with the given limb are finished the results are stored in two integer variables. The program now evaluates whether either of the angles are greater than 180 degrees. If not, it prints an ASCII string like “D035”, where D is a character representing the associated servo, and the next three digits represent the angle value the servo needs to position itself to. The output uses a USB based virtual COM port communicating at 4800 baud. Essentially, the program is sending a serial command every 10ms. The USB to RS232 device used is the ET-USB/RS232 Mini from ETT.co.ltd⁷. This device is used because it was readily at hand. The signal is then sent through a transistor inverter before going into the microcontroller input.

⁷ The software can be downloaded from this site
 [Accessed 04/28/2011] <<http://et-usb-rs232-mini.software.informer.com/2.0/>>

3.3.4 Communication Diagrams

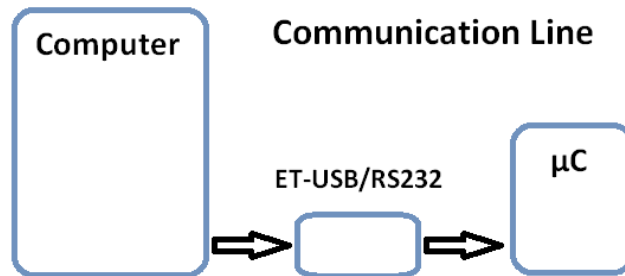


Figure 24 Communication Line

3.3.5 Translating Angle Data to PWM

The microcontroller used is an STM32F100RB on a STM32VL-Discovery development board. This controller was picked because of its low cost, speed, and the number of available timers. When data is sent to the microcontroller input, an interrupt condition occurs and the interrupt service routine takes in all the data and stores it for processing. When the interrupt is finished, the main program parses the data and then calls a function to output the data.

In order to create PWM⁸ outputs, each of the timers need to be configured. Eight PWM outputs can be obtained from Timer 1 and Timer 2, since both the timers have 4 Capture/Compare registers. To set the Timer configurations, the STM32 Standard Peripheral Library is used. First, the timer is set to run at 1MHz. With the period value set to 20000 counts, the output frequency is set to $1000000/20000 = 50\text{Hz}$. This is the standard servo frequency. Note that the system is actually working at 100Hz since the update rate from the computer is every 10ms. The original design was for 20ms, but from actual testing, the servos can run at 100Hz. The timer is set to an up counting mode where the output is high until the count reaches the value in the Capture/Compare register. At this point the output is low again until the timer reaches its entire period of 20ms, or in this case, until the Capture/Compare register is updated

⁸ PWM stands for pulse-width modulation

again after 10ms. Since the clock is running at 1MHz, each value in the Capture/Compare register corresponds to 1 μ s.

When the function to output the data is called, it converts the angle to a number corresponding to the number of microseconds that the output pulse width needs to be. From the servo data and test trials, the pulse width for 0 degrees is 556 μ s and the pulse width for 180 degrees is 2500 μ s. From this we can create the equation below for a linear conversion:

$$Pulsewidth = (Degree \times 10.88) + 556$$

After the pulse width is obtained, the Capture/Compare register is updated with this new value and the Timer automatically updates its PWM output, which is linked directly to an output pin via GPIO settings.

3.3.6 Output Circuitry

Servo Driver Circuit

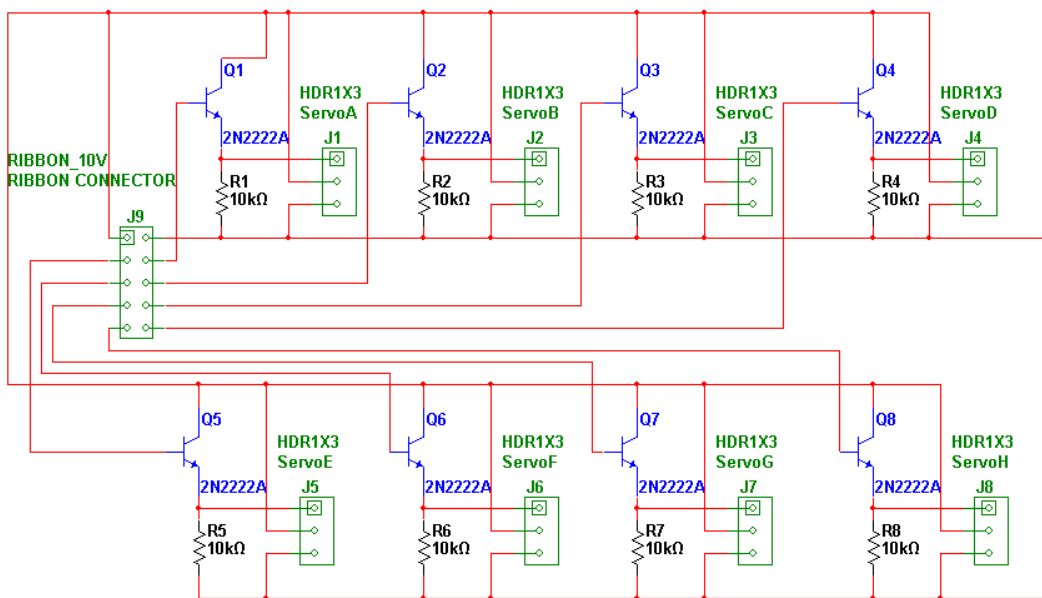


Figure 25 Servo Driver Circuit

4. Results and Discussion / Analysis

The Shadow Mirror is expected to recognize a user, track his or her actions, update visual puppet movement, and control the physical puppet. At the Boston Museum of Science we set up the display adjacent to a walking area. The space behind the user is a flat wall, which gave the motion capturing better performance and precision. We posed as users to interact with the shadow puppet and drew the attention of visitors to discover its capabilities. Numerous visitors stopped by and interacted with our shadow puppet and gave us feedback to help us understand more about the interactive experience.

Meanwhile, we also found potential problems and limits in the actual implementation of the motion tracking. For example, smaller children seemed to be more difficult to track because the Kinect was unable to determine the location of the knees and elbows in some cases. This resulted in a smaller scope of movement than expected and slightly hindered the user's experience. Some users also did relatively fast movements, and these motions were not always captured because they are difficult to track. The vision system could be enhanced to increase the performance of the motion capture for these specific situations.

Many users from different backgrounds, professional and non-professional, interacted with the Shadow Mirror. This allows us to better understand the different possibilities for future development and refinement of the project. The Robot Block Party held at Boston Museum of Science provided us this platform which gave us an opportunity to present our ideas, technologies, thoughts, and receive feedback from people who interacted with the shadow puppet.

5. Conclusions

The Shadow Mirror project embraced the goals we proposed. By participating in the Boston Museum of Science's Robot Block Party, our project was exposed to a wider range of audience. People in different of gender, age, culture enjoyed interacting with our shadow puppet. Universal friendship is

promoted via open communication between both the audience members and the creators. Through exchanging ideas with audience members who share the same interest with our project, we discovered more about the project's further possibilities, learned what other people are doing with the Kinect, and gained many ideas and insights on robots and puppets in general.

People found our shadow puppet to be visually appealing. A Japanese Byobu style screen veils the puppet and displays the puppet's shadow to audience. The mysterious shadow attracts people to discover what lies behind the screen; the bouncy movements infuse life in the moving shadow. Through the Shadow Mirror, we demonstrated that technology is not just for utilitarian purposes, but it can also be used as a medium for art

During the development of the Shadow Mirror, we encountered and overcame different obstacles. Some parts did not work out as expected, so we experimented and found better solutions. Everyone was challenged in applying their expertise. The project also required team cooperation, communication, and the establishment of expectations. This project has taught us to anticipate failure and better prepare ourselves for future endeavors. Overall, the creation of the Shadow Mirror was a delightful experience for both the creators and the audience.

6. Recommendations

Through hard testing with a wide variety of people at the Boston Museum of Science Robot Block Party, it was determined that there were a variety of different ways we could improve upon or enhance the design of The Shadow Mirror. First of all, when smaller children tried to use the puppet, the Kinect had difficulties tracking the child's knees and elbows, so control of the puppet had a few glitches. If there is a way to enhance the Kinect to adjust for this it would be recommended because children seemed especially interested in it, but their overall experience with the puppet was hindered because of this limitation. Another possible way to enhance the motion tracking is to resolve the issue of needing to

calibrate if too many people step in front of the camera at once. There were times when a large group of people would obstruct the Kinect's line of sight with the user for a period of a few seconds, resulting in a loss of the person's calibration. Though it did not dramatically affect the overall experience too much, it was still a slightly annoying situation when it occurred.

A larger range of motion for the puppet would greatly enhance the user experience as well. Allowing full control of the puppet's limbs would make for a much more involved participation of the user and give him or her more of a sense of control. This could be done using better motors, and a more developed tracking system to work with the Kinect. The addition of Z-direction movement and a larger shadow size could prove more involving as well, which could be achieved with a more advanced frame design and a dynamic track system for the servo motors.

Appendices

Appendix A: Part list and cost

Part	Quantity	Cost/Unit	Overall Cost
Computer	1	\$0.00	\$0.00
USB cables to connect to microcontroller	2	\$5.00	\$10.00
STM32VL-Discovery Development Board	1	\$12.00	\$12.00
Servo	9	\$10.00	\$90.00
Connectors for Servos to microcontroller	2	\$10.00	\$20.00
Power circuit for servos	1	\$0.00	\$0.00
Sports balls	5	\$2.00	\$10.00
Springs	5	\$0.00	\$0.00
Fishing Line	1	\$5.00	\$5.00
Wood for enclosure	1	\$10.00	\$10.00
Wood for wall	1	\$20.00	\$20.00
Byobu style screen	1	\$0.00	\$0.00
projector	1	\$0.00	\$0.00
Miscellaneous Parts			\$20.00
Microsoft Kinect for xBox	1	\$150.00	\$150.00
TOTAL			\$347.00

Appendix B: Pictures of Parts

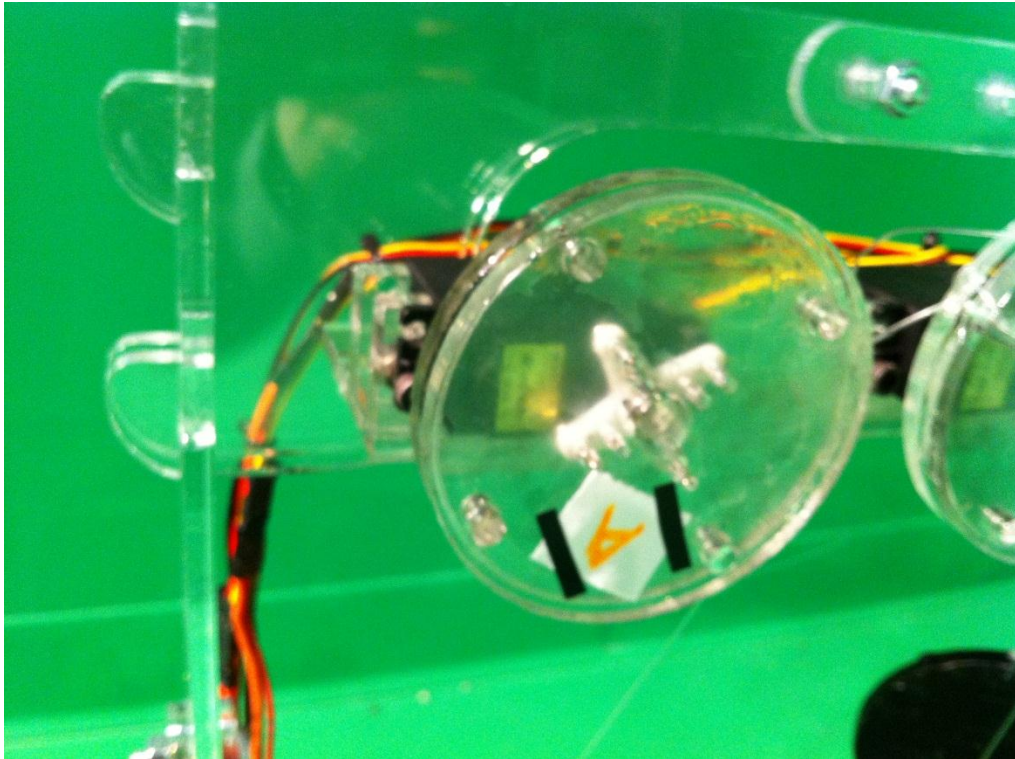


Figure 26 A pulley integrated onto a servo for better movements of the string attaché

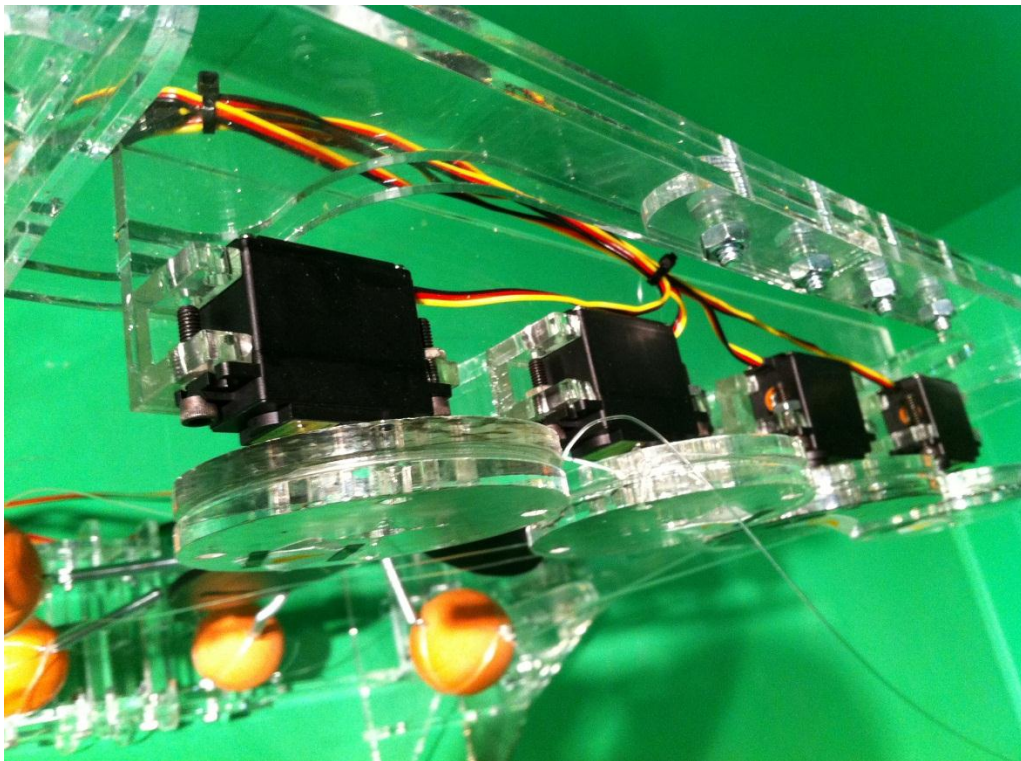


Figure 27 Back view of the top servos mounted on the puppet frame

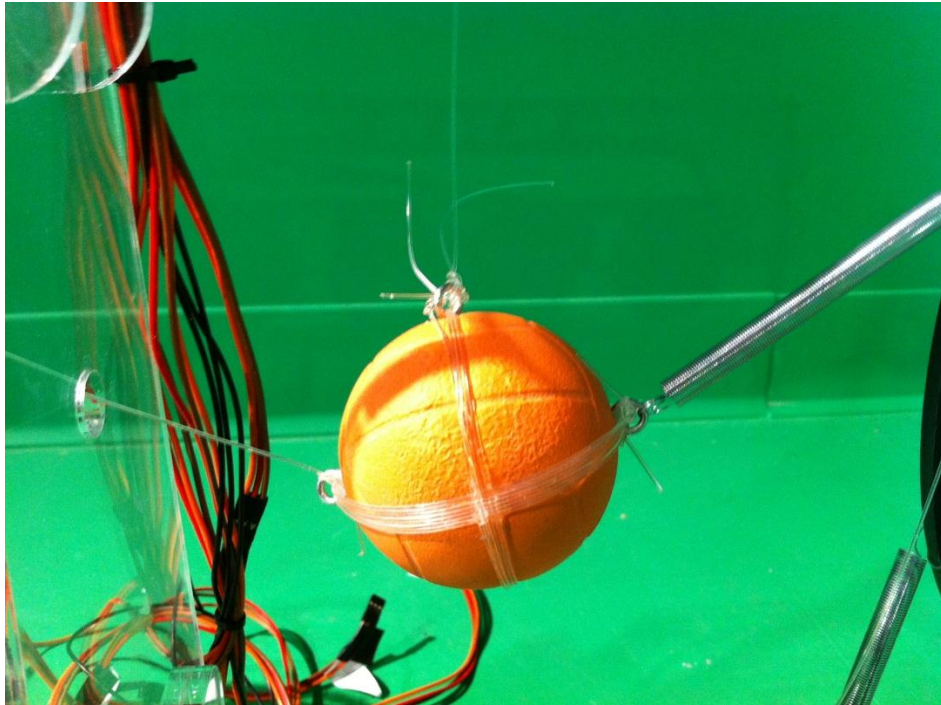


Figure 28 A ball representing a hand of the puppet, attached to a spring and 2 servo motors



Figure 29 An old Bell & Howell Project 'N' View 500 projector for the shadow projection of the puppet

Appendix C: Control System Code in C#

List of files

- Kinect Vision

UserTracker

main.cpp

SceneDrawer.cpp

SceneDrawer.h

OpenNI.h

- Control Program

Shadow_Puppet

form1.cs

program.cs

form1.cs

```
namespace IQP_ShadowPuppet_V1
{
    public partial class form_IQP_ShadowPuppet_V1 : Form
    {
        const int panel_width = 640;
        const int panel_height = 700;
        //Pulley radius
        double pulley_r = 56; //2 inches * 40(scalar) actual radius = 1.4 * 40(scalar)
        Servo ServoA = new Servo(); //right hand top
        Servo ServoB = new Servo(); //right hand side
        Servo ServoE = new Servo(); //left hand
        Servo ServoF = new Servo();
        Servo ServoC = new Servo(); //right foot
        Servo ServoD = new Servo();
        Servo ServoG = new Servo(); // left foot
        Servo ServoH = new Servo();
        System.Net.Sockets.TcpClient clientSocket = new System.Net.Sockets.TcpClient();
        TcpClient tcpclnt;
        NetworkStream stm;
        Point RightArmCenterLocation = new Point(400, 150);
        Point LeftArmCenterLocation = new Point(100, 150);
        Point RightLegCenterLocation = new Point(400, 400);
        Point LeftLegCenterLocation = new Point(100, 400);
    }
}
```

```

Point BodyCenterLocation = new Point(250, 250);
Point HeadCenterLocation = new Point(250, 50);
//String command send to servos
public void send(string cmd)
{
    byte[] asciiString = Encoding.ASCII.GetBytes(cmd);
    serialPort1.Write(asciiString, 0, asciiString.Length);
}
//Form constructor
public form_IQP_ShadowPuppet_V1()
{
    InitializeComponent();
    ServoA.name = "s_rh_top";
    ServoA.x = 540;
    ServoA.y = 0;
    ServoA.offset = 220;
    ServoB.name = "s_rh_side";
    ServoB.x = panel_width;
    ServoB.y = 420;
    ServoB.offset = 80;
    ServoE.name = "s_lh_top";
    ServoE.x = 100;
    ServoE.y = 0;
    ServoE.offset = 220;
    ServoF.name = "s_lh_side";
    ServoF.x = 0;
    ServoF.y = 420;
    ServoF.offset = 80;
    ServoC.name = "s_rf_top";
    ServoC.x = 390;
    ServoC.y = 0;
    ServoC.offset = 600;
    ServoD.name = "s_rf_side";
    ServoD.x = panel_width;
    ServoD.y = panel_height;
    ServoD.offset = 100;
    ServoG.name = "s_lf_top";
    ServoG.x = 250;
    ServoG.y = 0;
    ServoG.offset = 600;
    ServoH.name = "s_lf_side";
    ServoH.x = 0;
    ServoH.y = panel_height;
    ServoH.offset = 100;
}
// UPDATE ACTUAL CHARACTER (SHOULD BE TIMER BASED)
private void timer1_Tick(object sender, EventArgs e){
    try{
        byte[] bb = new byte[1024];
        int k = 0;
        if (stm.CanRead)
        {
            k = stm.Read(bb, 0, bb.Length);
            string data = System.Text.Encoding.ASCII.GetString(bb);
            label1.Text = data;
        }
    }
}

```

```

string temp = label1.Text;
Pkt newp = new Pkt();
newp = parse(temp);

XY moveTo = new XY();
moveTo = norm(newp);
if (newp.name == "rh")
{
    Point newP = new Point(this.pictureBox_Body.Location.X + newp.xb, this.pictureBox_Body.Location.Y -
newp.yb);

    int m = (ServoA.y - ServoB.y) / (ServoA.x - ServoB.x);
    int b = ServoA.y - m * ServoA.x;
    if (newP.Y > (m * newP.X + b)){
        this.pictureBox_RightHand.Location = newP;
        XY locA = new XY();
        locA.nx = this.ServoA.x;
        locA.ny = this.ServoA.y;
        XY locB = new XY();
        locB.nx = this.ServoB.x;
        locB.ny = this.ServoB.y;
        XY point = new XY();
        point.nx = this.pictureBox_RightHand.Location.X;
        point.ny = this.pictureBox_RightHand.Location.Y;
        int angle;
        angle = this.find_servo_A_angle(locA, point);
        int angle2;
        angle2 = this.find_servo_B_angle(locB, point);
        if (angle <= 180 && angle2 <=180) {
            this.send("A" + angle.ToString());
            this.send("B" + angle2.ToString());
            angle = angle2 = 0;
        }
    } // end of inside boundary
}
else if (newp.name == "lh")
{
    Point newP = new Point(this.pictureBox_Body.Location.X + newp.xb, this.pictureBox_Body.Location.Y -
newp.yb)

    int m = (ServoE.y - ServoF.y) / (ServoE.x - ServoF.x);
    int b = ServoE.y - m * ServoE.x;
    if (newP.Y > (m * newP.X + b))
    {
        this.pictureBox_LeftHand.Location = newP;
        XY locE = new XY();
        locE.nx = this.ServoE.x;
        locE.ny = this.ServoE.y;
        XY locF = new XY();
        locF.nx = this.ServoF.x;
        locF.ny = this.ServoF.y;
        XY point = new XY();
        point.nx = this.pictureBox_LeftHand.Location.X;
        point.ny = this.pictureBox_LeftHand.Location.Y;
        int angle;
        angle = 180 - this.find_servo_E_angle(locE, point);
        int angle2;
        angle2 = 180 - this.find_servo_F_angle(locF, point);
    }
}

```

```

        if (angle <= 180 && angle2 <= 180)
        {
            this.send("E" + angle.ToString());
            this.send("F" + angle2.ToString());
        }

    } // end of inside boundary
}
else if (newp.name == "rf")
{
    Point newP = new Point(this.pictureBox_Body.Location.X + newp.xb, this.pictureBox_Body.Location.Y -
newp.yb);
    int m = (ServoC.y - ServoD.y) / (ServoC.x -
ServoD.x);
    int b = ServoC.y - m * ServoC.x;
    if (newP.Y > (m * newP.X + b))
    {
        this.pictureBox_RightFoot.Location = newP;
        XY locC = new XY();
        locC.nx = this.ServoC.x;
        locC.ny = this.ServoC.y;
        XY locD = new XY();
        locD.nx = this.ServoD.x;
        locD.ny = this.ServoD.y;
        XY point = new XY();
        point.nx = this.pictureBox_RightFoot.Location.X;
        point.ny = this.pictureBox_RightFoot.Location.Y;
        int angle;
        angle = this.find_servo_C_angle(locC, point);
        int angle2;
        angle2 = this.find_servo_D_angle(locD,
point);

        if(angle <=180 && angle2 <=180){
            this.send("C" + angle.ToString());
            this.send("D" + angle2.ToString());
        }
    } // end of inside boundary
}
else if (newp.name == "lf")
{
    Point newP = new Point(this.pictureBox_Body.Location.X + newp.xb, this.pictureBox_Body.Location.Y -
newp.yb);
    int m = (ServoG.y - ServoH.y) / (ServoG.x - ServoH.x);
    int b = ServoG.y - m * ServoG.x;
    if (newP.Y > (m * newP.X + b))
    {
        this.pictureBox_LeftFoot.Location = newP;
        XY locG = new XY();
        locG.nx = this.ServoG.x;
        locG.ny = this.ServoG.y;
        XY locH = new XY();
        locH.nx = this.ServoH.x;
        locH.ny = this.ServoH.y;
        XY point = new XY();
        point.nx = this.pictureBox_LeftFoot.Location.X;
        point.ny = this.pictureBox_LeftFoot.Location.Y;
    }
}
}
}

```



```

tcpclnt = new TcpClient();
label1.Text = "Connecting.....";
tcpclnt.Connect("127.0.0.1", 51006);
label1.Text = "Client Socket Program - Server Connected ...";
stm = tcpclnt.GetStream();
byte[] bb = new byte[1024];
for (int k = 0; k < 1; k++)
{
    if (stm.CanRead)
    {
        stm.Read(bb, 0, 1024);
        string returndata = System.Text.Encoding.ASCII.GetString(bb);
    }
    else {
        msg("cannot read!\n");
    }
}
serialPort1.Open();
}
catch (Exception e1)
{
    msg("Error..... " + e1.StackTrace);
}
}
public void msg(string mesg)
{
    label1.Text = Environment.NewLine + ">>> " + mesg;
}
//Servo A angle, right hand top
public int find_servo_A_angle(XY loc, XY a_point) {
    int angle_A;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoA.offset;
    angle_A = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_A;
}
//Servo B angle right hand side
public int find_servo_B_angle(XY loc, XY b_point)
{
    int angle_B;
    double length;
    length = Math.Sqrt(Math.Pow(b_point.nx - loc.nx, 2) + Math.Pow(b_point.ny - loc.ny, 2)) - this.ServoB.offset;
    angle_B = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_B;
}
//Servo E angle, left hand top
public int find_servo_E_angle(XY loc, XY a_point)
{
    int angle_E;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoE.offset;
    angle_E = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_E;
}
//Servo F angle, left hand side

```

```

public int find_servo_F_angle(XY loc, XY a_point)
{
    int angle_F;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoF.offset;
    angle_F = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_F;
}
//Servo C angle, right hand side
public int find_servo_C_angle(XY loc, XY a_point)
{
    int angle_C;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoC.offset;
    angle_C = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_C;
}
//Servo D angle, right hand side
public int find_servo_D_angle(XY loc, XY a_point)
{
    int angle_D;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoD.offset;
    angle_D = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_D;
}
//Servo G angle, left hand side
public int find_servo_G_angle(XY loc, XY a_point)
{
    int angle_G;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoG.offset;
    angle_G = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_G;
}
//Servo H angle, left hand side
public int find_servo_H_angle(XY loc, XY a_point)
{
    int angle_H;
    double length;
    length = Math.Sqrt(Math.Pow(a_point.nx - loc.nx, 2) + Math.Pow(a_point.ny - loc.ny, 2)) - this.ServoH.offset;
    angle_H = (int)((length / (pulley_r * Math.PI)) * 180);
    return angle_H;
}
//helper parsing received coord values
public Pkt parse(string msg)
{
    Pkt pk = new Pkt();
    string[] words = msg.Split(',');
    for (int i = 0; i < 4; i++)
    {
        if(i ==0){
            pk.xb =Convert.ToInt32(((string)words[i]).Trim());
        }if(i ==1){
            pk.yb =Convert.ToInt32(((string)words[i]).Trim());
        }
    }
}

```

```

    }
    if (i == 3) { pk.name = (string)words[i];}
    }
    return pk;
}
//helper that mapping to the form window
public XY norm(Pkt pk)
{
    XY newxy = new XY();
    int ww = 720, wh = 480; //Kinect window dimension
    int vw = 900, vh = 788; //Puppet window dimension
    newxy.nx = pk.xb;
    newxy.ny = pk.yb;
    return newxy;
}
public int convertToPixel(double inch)
{
    int pixel;
    pixel = (int)inch * 40 / 1;
    return pixel;
}
private void form_IQP_ShadowPuppet_V1_FormClosing(object sender, FormClosingEventArgs e)
{
    serialPort1.Close();
}
}
}

```


Appendix D: STM32 Code

Main.c

```
/**
*****
* @file main.c
* @author Warranty Walton
* @date 4/2/2011
* @brief Main program body
*****
**/

/* Includes -----*/

#include "stm32f10x.h"
#include "global.h"
#include "stdlib.h"
// #include "stm32_eval.h"

/* Private typedef -----*/
/* Private define -----*/
/* Private macro -----*/
/* Private variables -----*/

int Degree;
uint8_t DesignatedServo;
uint8_t InputBuffer[4];
uint8_t ReadyToParse;
unsigned int ServoOut;
USART_InitTypeDef USART_InitStructure;

/* Private function prototypes -----*/

void NVIC_Configuration(void);
void COMInit(USART_InitTypeDef* USART_InitStructure);
void parseInput(void);
uint16_t convertToTimerCCRValue(int Degree);

/* Private functions -----*/

/**
* @brief Main program
**/

int main(void)
{
    /* NVIC configuration */
    NVIC_Configuration();

    /* USARTx configured as follow:
    - BaudRate = 9600 baud
    - Word Length = 8 Bits
    - One Stop Bit
    - No parity
    - Hardware flow control disabled (RTS and CTS signals)
    */
}
```

```

- Receive and transmit enabled
*/
USART_InitStructure.USART_BaudRate = 4800;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

COMInit(&USART_InitStructure);

/* Enable the USART1 Transmit interrupt: this interrupt is generated when the
   USART1 transmit data register is empty */

USART_ITConfig(USART1, USART_IT_TXE, ENABLE);

/* Enable the USART1 Receive interrupt: this interrupt is generated when the
   USART1 receive data register is not empty */

USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

/* RCC, GPIO, and TIMER configuration */

TIMER_RCC_Configuration();
TIMER_GPIO_Configuration();
TIM2_Configuration();
TIM3_Configuration();
TIM_Cmd(TIM2, ENABLE);
TIM_Cmd(TIM3, ENABLE);

while (1)
{
    /* Parse Input */
    if(ReadyToParse)
    {
        // Disable Receive Interrupt so that the parsing is not corrupted
        USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
        parseInput();
        ReadyToParse = 0;
        // Enable Receive Interrupt
        USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    }

    /* Servo Select */
    int i;
    switch(DesignatedServo)
    {
    case 'A':
        OutputToServo_A(convertToTimerCCRValue(Degree));
        DesignatedServo = 0;
        for(i=0;i<4;i++)
        {
            InputBuffer[i] = 0;
        }
        break;
    case 'B':
        OutputToServo_B(convertToTimerCCRValue(Degree));
        DesignatedServo = 0;
    }
}

```

```

        for(i=0;i<4;i++)
        {
            InputBuffer[i] = 0;
        }
        break;
case 'C':
    OutputToServo_C(convertToTimerCCRValue(Degree));
    DesignatedServo = 0;
    for(i=0;i<4;i++)
    {
        InputBuffer[i] = 0;
    }
    break;
case 'D':
    OutputToServo_D(convertToTimerCCRValue(Degree));
    DesignatedServo = 0;
    for(i=0;i<4;i++)
    {
        InputBuffer[i] = 0;
    }
    break;
case 'E':
    OutputToServo_E(convertToTimerCCRValue(Degree));
    DesignatedServo = 0;
    for(i=0;i<4;i++)
    {
        InputBuffer[i] = 0;
    }
    break;
case 'F':
    OutputToServo_F(convertToTimerCCRValue(Degree));
    DesignatedServo = 0;
    for(i=0;i<4;i++)
    {
        InputBuffer[i] = 0;
    }
    break;
case 'G':
    OutputToServo_G(convertToTimerCCRValue(Degree));
    DesignatedServo = 0;
    for(i=0;i<4;i++)
    {
        InputBuffer[i] = 0;
    }
    break;
case 'H':
    OutputToServo_H(convertToTimerCCRValue(Degree));
    DesignatedServo = 0;
    for(i=0;i<4;i++)
    {
        InputBuffer[i] = 0;
    }
    break;
}
}
}

```

```

/**
 * @brief Configures the nested vectored interrupt controller.
 ***/

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Enable the USART1 Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

/**
 * @brief Initializes the USART and associated IO's
 ***/

void COMInit(USART_InitTypeDef* USART_InitStructure)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /* Enable GPIO clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);

    /* Enable UART clock */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    /* Configure USART Tx as alternate function push-pull */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure USART Rx as input floating */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* USART configuration */
    USART_Init(USART1, USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART1, ENABLE);
}

/**
 * @brief Parses the input and puts the resulting integer in "Degree"
 ***/

void parseInput(void)
{
    uint8_t DegreeArray[3];
    DesignatedServo = InputBuffer[0];
    int i = 0;
}

```

```

        for(i=0; i<3; i++)
        {
            DegreeArray[i] = InputBuffer[i+1];
        }
        Degree = atoi(DegreeArray);
        ServoOut = convertToTimerCCRValue(Degree);
    }

/**
 * @brief Converts degrees into appropriate Capture/Compare Register values (microseconds)
 ***/

uint16_t convertToTimerCCRValue(int Degree)
{
    uint16_t result;
    if(Degree > 180)
    {
        Degree = 180;
    }
    else if(Degree < 0)
    {
        Degree = 0;
    }
    result = (uint16_t) (((float)Degree*(10.8)) + 556);
    return result;
}

/*****END OF FILE*****/

```

Global.h

```

/**
 ****
 * @file global.h
 * @author Warranyu Walton
 * @date 4/2/2011
 * @brief global functions and variables
 ****
 **/

/* Public variables -----*/

extern uint8_t InputBuffer[4];
extern uint8_t ReadyToParse;

/* Public function prototypes -----*/

void TIM2_Configuration(void);
void TIM3_Configuration(void);
void TIM4_Configuration(void);
void TIM5_Configuration(void);
void TIMER_RCC_Configuration(void);

```

```

void TIMER_GPIO_Configuration(void);
void OutputToServo_A(uint16_t servo_output);
void OutputToServo_B(uint16_t servo_output);
void OutputToServo_C(uint16_t servo_output);
void OutputToServo_D(uint16_t servo_output);
void OutputToServo_E(uint16_t servo_output);
void OutputToServo_F(uint16_t servo_output);
void OutputToServo_G(uint16_t servo_output);
void OutputToServo_H(uint16_t servo_output);

```

```

/*****END OF FILE*****/

```

Global.c

```

/**
*****
 * @file global.c
 * @author Warranyu Walton
 * @date 4/2/2011
 * @brief global functions and variables
*****
**/

/* Includes -----*/

#include "stm32f10x.h"

/* Private typedef -----*/

TIM_TimeBaseInitTypeDef TIM2_TimeBaseStructure;
TIM_TimeBaseInitTypeDef TIM3_TimeBaseStructure;
TIM_TimeBaseInitTypeDef TIM4_TimeBaseStructure;
TIM_TimeBaseInitTypeDef TIM5_TimeBaseStructure;
TIM_OCInitTypeDef TIM2_OCInitStructure;
TIM_OCInitTypeDef TIM3_OCInitStructure;
TIM_OCInitTypeDef TIM4_OCInitStructure;
TIM_OCInitTypeDef TIM5_OCInitStructure;

/* Private define -----*/
/* Private macro -----*/
/* Private variables -----*/

uint16_t PrescalerValue = 0;

/* Public functions -----*/

/**
 * @brief Enable timer clocks and related IO's
***/

void TIMER_RCC_Configuration(void)
{
    /* TIM2-5 clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2 | RCC_APB1Periph_TIM3 | RCC_APB1Periph_TIM4 |
RCC_APB1Periph_TIM5, ENABLE);

```

```

        /* GPIOA, GPIOB, and GPIOC clock enable */
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
            RCC_APB2Periph_GPIOC | RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO, ENABLE);
    }

    /**
     * @brief Timer GPIO speed and mode configuration.
     ***/

void TIMER_GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /* GPIOA Configuration:TIM3 Channel1, 2, 3 and 4 as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_6 | GPIO_Pin_7;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure USART1 Rx as input floating */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure USART1 Tx as alternate function push-pull */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* GPIOB Config */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
}

    /**
     * @brief Configures TIMER2's prescaler value, mode, and period.
     ***/

void TIM2_Configuration(void)
{
    /* -----
    TIM2 Configuration: generate 4 PWM signals with 4 different duty cycles:
    The TIM2CLK frequency is set to SystemCoreClock (Hz), to get TIM2 counter
    clock at 1 MHz the Prescaler is computed as following:
    - Prescaler = (TIM2CLK / TIM2 counter clock) - 1
    SystemCoreClock is set to 24 MHz for Low-Density Value line and
    Medium-Density Value line devices

    The TIM2 is running at 50 Hz: TIM2 Frequency = TIM2 counter clock/(ARR + 1)
    = 1 MHz / 20000 = 50 Hz
    ----- */
    /* Compute the prescaler value */
    PrescalerValue = (uint16_t) (SystemCoreClock / 1000000) - 1;
}

```

```

/* Time base configuration */
TIM2_TimeBaseStructure.TIM_Period = 19999;
TIM2_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM2_TimeBaseStructure.TIM_ClockDivision = 0;
TIM2_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM2, &TIM2_TimeBaseStructure);

/* PWM1 Mode configuration: Channel1 */
TIM2_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM2_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM2_OCInitStructure.TIM_Pulse = 0;
TIM2_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OC1Init(TIM2, &TIM2_OCInitStructure);

TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel2 */
TIM2_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM2_OCInitStructure.TIM_Pulse = 0;

TIM_OC2Init(TIM2, &TIM2_OCInitStructure);

TIM_OC2PreloadConfig(TIM2, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel3 */
TIM2_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM2_OCInitStructure.TIM_Pulse = 0;

TIM_OC3Init(TIM2, &TIM2_OCInitStructure);

TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel4 */
TIM2_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM2_OCInitStructure.TIM_Pulse = 0;

TIM_OC4Init(TIM2, &TIM2_OCInitStructure);

TIM_OC4PreloadConfig(TIM2, TIM_OCPreload_Enable);

TIM_ARRPreloadConfig(TIM2, ENABLE);
}

/**
 * @brief Configures TIM3's prescaler value, mode, and period.
 */

void TIM3_Configuration(void)
{
    /* -----
    TIM3 Configuration: generate 4 PWM signals with 4 different duty cycles:
    The TIM3CLK frequency is set to SystemCoreClock (Hz), to get TIM3 counter
    clock at 1 MHz the Prescaler is computed as following:
    - Prescaler = (TIM3CLK / TIM3 counter clock) - 1
    SystemCoreClock is set to 24 MHz for Low-Density Value line and
    Medium-Density Value line devices
    */
}

```


The TIM3 is running at 50 Hz: TIM3 Frequency = TIM3 counter clock/(ARR + 1)
= 1 MHz / 20000 = 50 Hz

TIM3 Channel1 duty cycle = (TIM3_CCR1/ TIM3_ARR)* 100 = 50%
TIM3 Channel2 duty cycle = (TIM3_CCR2/ TIM3_ARR)* 100 = 37.5%
TIM3 Channel3 duty cycle = (TIM3_CCR3/ TIM3_ARR)* 100 = 25%
TIM3 Channel4 duty cycle = (TIM3_CCR4/ TIM3_ARR)* 100 = 12.5%

```
----- */
/* Compute the prescaler value */
PrescalerValue = (uint16_t) (SystemCoreClock / 1000000) - 1;
/* Time base configuration */
TIM3_TimeBaseStructure.TIM_Period = 19999;
TIM3_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM3_TimeBaseStructure.TIM_ClockDivision = 0;
TIM3_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

TIM_TimeBaseInit(TIM3, &TIM3_TimeBaseStructure);

/* PWM1 Mode configuration: Channel1 */
TIM3_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM3_OCInitStructure.TIM_Pulse = 0;
TIM3_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OC1Init(TIM3, &TIM3_OCInitStructure);

TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel2 */
TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM3_OCInitStructure.TIM_Pulse = 0;

TIM_OC2Init(TIM3, &TIM3_OCInitStructure);

TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel3 */
TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM3_OCInitStructure.TIM_Pulse = 0;

TIM_OC3Init(TIM3, &TIM3_OCInitStructure);

TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel4 */
TIM3_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM3_OCInitStructure.TIM_Pulse = 0;

TIM_OC4Init(TIM3, &TIM3_OCInitStructure);

TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);

TIM_ARRPreloadConfig(TIM3, ENABLE);
}

/**
 * @brief Configures TIM3's prescaler value, mode, and period.
 ***/
```

```

void TIM4_Configuration()
{
    /* -----
    TIM4 Configuration: generate 4 PWM signals with 4 different duty cycles:
    The TIM4CLK frequency is set to SystemCoreClock (Hz), to get TIM4 counter
    clock at 1 MHz the Prescaler is computed as following:
    - Prescaler = (TIM4CLK / TIM4 counter clock) - 1
    SystemCoreClock is set to 24 MHz for Low-Density Value line and
    Medium-Density Value line devices

    The TIM4 is running at 50 Hz: TIM4 Frequency = TIM4 counter clock/(ARR + 1)
    = 1 MHz / 20000 = 50 Hz
    TIM3 Channel1 duty cycle = (TIM3_CCR1/ TIM3_ARR)* 100 = 50%
    TIM3 Channel2 duty cycle = (TIM3_CCR2/ TIM3_ARR)* 100 = 37.5%
    TIM3 Channel3 duty cycle = (TIM3_CCR3/ TIM3_ARR)* 100 = 25%
    TIM3 Channel4 duty cycle = (TIM3_CCR4/ TIM3_ARR)* 100 = 12.5%
    ----- */
    /* Compute the prescaler value */
    PrescalerValue = (uint16_t) (SystemCoreClock / 1000000) - 1;
    /* Time base configuration */
    TIM4_TimeBaseStructure.TIM_Period = 19999;
    TIM4_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
    TIM4_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM4_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM4, &TIM4_TimeBaseStructure);

    /* PWM1 Mode configuration: Channel1 */
    TIM4_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM4_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM4_OCInitStructure.TIM_Pulse = 0;
    TIM4_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

    TIM_OC1Init(TIM4, &TIM4_OCInitStructure);

    TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel2 */
    TIM4_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM4_OCInitStructure.TIM_Pulse = 0;

    TIM_OC2Init(TIM4, &TIM4_OCInitStructure);

    TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel3 */
    TIM4_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM4_OCInitStructure.TIM_Pulse = 0;

    TIM_OC3Init(TIM4, &TIM4_OCInitStructure);

    TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel4 */
    TIM4_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM4_OCInitStructure.TIM_Pulse = 0;

```

```

TIM_OC4Init(TIM4, &TIM4_OCInitStructure);

TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);

TIM_ARRPreloadConfig(TIM4, ENABLE);
}

/**
 * @brief Configures TIMERS's prescaler value, mode, and period.
 ***/

void TIM5_Configuration(void)
{
    /* -----
    TIM5 Configuration: generate 4 PWM signals with 4 different duty cycles:
    The TIM5CLK frequency is set to SystemCoreClock (Hz), to get TIM5 counter
    clock at 1 MHz the Prescaler is computed as following:
    - Prescaler = (TIM5CLK / TIM5 counter clock) - 1
    SystemCoreClock is set to 24 MHz for Low-Density Value line and
    Medium-Density Value line devices

    The TIM5 is running at 50 Hz: TIM5 Frequency = TIM5 counter clock/(ARR + 1)
    = 1 MHz / 20000 = 50 Hz
    ----- */
    /* Compute the prescaler value */
    PrescalerValue = (uint16_t) (SystemCoreClock / 1000000) - 1;
    /* Time base configuration */
    TIM5_TimeBaseStructure.TIM_Period = 19999;
    TIM5_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
    TIM5_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM5_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM5, &TIM5_TimeBaseStructure);

    /* PWM1 Mode configuration: Channel1 */
    TIM5_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    TIM5_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM5_OCInitStructure.TIM_Pulse = 0;
    TIM5_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

    TIM_OC1Init(TIM5, &TIM5_OCInitStructure);

    TIM_OC1PreloadConfig(TIM5, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel2 */
    TIM5_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM5_OCInitStructure.TIM_Pulse = 0;

    TIM_OC2Init(TIM5, &TIM5_OCInitStructure);

    TIM_OC2PreloadConfig(TIM5, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel3 */
    TIM5_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM5_OCInitStructure.TIM_Pulse = 0;

    TIM_OC3Init(TIM5, &TIM5_OCInitStructure);

```

```

    TIM_OC3PreloadConfig(TIM5, TIM_OCPreload_Enable);

    /* PWM1 Mode configuration: Channel4 */
    TIM5_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM5_OCInitStructure.TIM_Pulse = 0;

    TIM_OC4Init(TIM5, &TIM5_OCInitStructure);

    TIM_OC4PreloadConfig(TIM5, TIM_OCPreload_Enable);

    TIM_ARRPreloadConfig(TIM5, ENABLE);
}

/**
 * @brief Updates TIMER2's CCR1
 */

void OutputToServo_A(uint16_t servo_output)
{
    TIM2_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC1Init(TIM2, &TIM2_OCInitStructure);
}

/**
 * @brief Updates TIMER2's CCR2
 */

void OutputToServo_B(uint16_t servo_output)
{
    TIM2_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC2Init(TIM2, &TIM2_OCInitStructure);
}

/**
 * @brief Updates TIMER2's CCR3
 */

void OutputToServo_C(uint16_t servo_output)
{
    TIM2_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC3Init(TIM2, &TIM2_OCInitStructure);
}

/**
 * @brief Updates TIMER2's CCR4
 */

void OutputToServo_D(uint16_t servo_output)
{
    TIM2_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC4Init(TIM2, &TIM2_OCInitStructure);
}

/**
 * @brief Updates TIMER3's CCR1
 */

void OutputToServo_E(uint16_t servo_output)

```

```

{
    TIM3_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC1Init(TIM3, &TIM3_OCInitStructure);
}

/**
 * @brief Updates TIMER3's CCR2
 */

void OutputToServo_F(uint16_t servo_output)
{
    TIM3_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC2Init(TIM3, &TIM3_OCInitStructure);
}

/**
 * @brief Updates TIMER3's CCR3
 */

void OutputToServo_G(uint16_t servo_output)
{
    TIM3_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC3Init(TIM3, &TIM3_OCInitStructure);
}

/**
 * @brief Updates TIMER3's CCR4
 */

void OutputToServo_H(uint16_t servo_output)
{
    TIM3_OCInitStructure.TIM_Pulse = servo_output;
    TIM_OC4Init(TIM3, &TIM3_OCInitStructure);
}

/*****END OF FILE*****/

```

stm32f10x_it.c

```

/**
*****
 * @file stm32f10x_it.c
 * @author MCD Application Team
 * @version V3.4.0
 * @date 10/15/2010
 * @brief Main Interrupt Service Routines.
 * This file provides template for all exceptions handler and peripherals
 * interrupt service routine.
*****
 * @copy
 *
 * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
 * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
 * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
 * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING

```

```

* FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
* CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
*
* <h2><center>&copy; COPYRIGHT 2010 STMicroelectronics</center></h2>
*/

/* Includes -----*/
#include "stm32f10x_it.h"
#include "global.h"
// #include "stm32_eval.h"

/** @addtogroup STM32F10x_StdPeriph_Examples
 * @{
 */

/** @addtogroup USART_HyperTerminal_Interrupt
 * @{
 */

/* Private typedef -----*/
/* Private define -----*/

#define TxBufferSize  0x04
#define RxBufferSize  0x04

/* Private macro -----*/
#define countof(a)  (sizeof(a) / sizeof(*(a)))

/* Private variables -----*/
uint8_t TxBuffer[] = "0000";
uint8_t RxBuffer[RxBufferSize];
uint8_t NbrOfDataToTransfer = TxBufferSize;
uint8_t NbrOfDataToRead = RxBufferSize;
uint8_t TxCounter = 0;
uint16_t RxCounter = 0;

/* Private function prototypes -----*/
/* Private functions -----*/

/*****
 * STM32F10x Peripherals Interrupt Handlers
 *****/

/**
 * @brief This function handles USARTx global interrupt request.
 * @param None
 * @retval None
 */
void USART1_IRQHandler(void)
{
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        /* Read one byte from the receive data register */
        RxBuffer[RxCounter++] = (USART_ReceiveData(USART1) & 0x7F);
        if((RxBuffer[0] == 'A') | (RxBuffer[0] == 'B') | (RxBuffer[0] == 'C') | (RxBuffer[0] == 'D') | (RxBuffer[0] == 'E') | (RxBuffer[0] ==
'F') | (RxBuffer[0] == 'G') | (RxBuffer[0] == 'H') | (RxBuffer[0] == 'I') | (RxBuffer[0] == 'J'))
        {
            if(RxCounter >= NbrOfDataToRead)

```

```

        {
            /* Disable the USART1 Receive interrupt */
            USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
            int i;
            for(i=0; i<RxCounter; i++)
            {
                InputBuffer[i] = RxBuffer[i];
                //TxBuffer[i] = RxBuffer[i];
            }
            RxCounter = 0;
            ReadyToParse = 1;
            /* Enable USART1 Transmit Interrupt */
            //USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
        }
    }
else
{
    RxBuffer[0] = 0;
    RxCounter = 0;
    ReadyToParse = 0;
}
}

if(USART_GetITStatus(USART1, USART_IT_TXE) != RESET)
{
    /* Write one byte to the transmit data register */
    USART_SendData(USART1, TxBuffer[TxCounter++]);

    if(TxCounter >= NbrOfDataToTransfer)
    {
        TxCounter = 0;
        /* Disable the USART1 Transmit interrupt */
        USART_ITConfig(USART1, USART_IT_TXE, DISABLE);

        /* Enable USART1 Receive Interrupt */
        USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    }
}
}

/***** (C) COPYRIGHT 2010 STMicroelectronics *****/

```

References

- [1] Kac, Eduardo. Foundation and Development of Robotic Art. *Art Journal*. Vol. 56, No. 3, Digital Reflections: The Dialogue of Art and Technology (Autumn, 1997), pp. 60-67
- [2] D. Currell, Puppets and Puppet Theatre. Wiltshire, UK: Crowood Press, 1999.
- [3] GROSS, Joan. Speaking in Other Voices: An ethnography of Walloon puppet theaters. Philadelphia, PA, USA: John Benjamins Publishing Company, 2001
- [4] Chinese Piyinxi, 2008. [Accessed 10/20/2010] <<http://www.u148.net/article/1624.html>>
- [5] Macmurtrie, Chico. Amorphic Robot Works. [Accessed 10/15/2010] <<http://amorphicrobotworks.org/works/ftm/index.htm>>
- [6] Joyce, Susan. The Fish Boy's Dream. January 21, 2006. <<http://srl.org/shows/la2006/>>
- [7] Martin, P; Egerstedt, M. Optimal Timing Control of Interconnected, Switched Systems with Applications to Robotic Marionettes. [Accessed 10/15/2010] <<http://users.ece.gatech.edu/~magnus/Papers/PuppetWODES08.pdf>>
- [8] Robotic Marionette Systems. 2006. <<http://155.69.254.10/users/risc/www/enter-intro.html>>
- [9] Levin, Golan. Ghost Pole Propagator. 2007. <<http://www.flong.com/projects/gpp/>>
- [10] Biggs, Simon. Shadows: An Interactive Digital Video Projection Installation. 1993. <<http://www.medienkunstnetz.de/works/shadows/images/3/>>
- [11] Rozin, Daniel. Wooden Mirror. 1999. <http://www.smoothware.com/danny/woodenmirror.html>
- [12] David Rockeby. Transforming Mirror. 1996. <<http://homepage.mac.com/davidrokeby/mirrorsmirrors.html>>