Design and Control of a Mobile Light Ballet Platform

by

Joshua Vincent Prescott Sibblies

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science

at the

Massachusetts Institute of Technology

June 2014

©2014 Joshua V.P. Sibblies. All rights reserved.

The author hereby grants to MIT permission to reproduce
and to distribute publicly paper and electronic
copies of this thesis document in whole or in part
in any medium now known or hereafter created.

Signature redacted

Department of Mechanical Engineering
May 10, 2010

Signature redacted

David L. Trumper
Professor of Mechanical Engineering
Thesis Supervisor

Annette Hosoi
Associate Professor of Mechanical Engineering
Undergraduate Officer
Control and Design of a Mobile Light Ballet Platform

by

Joshua Vincent Prescott Sibblies

Submitted to the Department of Mechanical Engineering on May 9, 2014
in partial fulfillment of the requirements for the degree of
Bachelor of Science in Mechanical Engineering

Abstract

Light ballet is the merging of art and technology; halogen lamps mounted on a turntable are placed inside of an opaque, perforated box. Light escapes through the perforations and creates a display of lights on surrounding surfaces. This thesis documents the process of transforming the generally stationary art display into a mobile exhibit. The light ballet modules will be placed on a mobile platform driven by three omni-directional wheels. The platform is controlled by an Android application, which communicates through Bluetooth and an Arduino Uno with the motor's controllers. The platform is designed to operate either through batteries, or through a retractable tether plugged into a power source, so the potential operating time is indefinite. This design will be replicated, to create a fleet of three light ballet robots for performance in Berlin, Germany, August 2014.

Thesis Supervisor: David L. Trumper
Title: Professor of Mechanical Engineering
Acknowledgements

I would like to thank all of the people who made working on and finishing this project possible.

To Professor David Trumper, thank you for helping me find this project, for being so supportive as I struggled through it, and for spending so much time helping me edit the final paper. I couldn’t have done it without you.

To Allan Doyle and Seth Riskin, thank you for your constant help and guidance as I worked though the project. I literally could not have gotten this far without all of your help.

To John Powell, thank you for being such a huge help with the fabrication of the new frames, and for all the help double checking my designs over the past three months.

To Steven Jorgensen and Bee Vang, thank you for all of your work and fabulous documentation on previous iterations of the project. It provided a fantastic base to work off of.
# Table of Contents

Abstract ................................................................................................................................. 3  
Acknowledgements .............................................................................................................. 4  
List of Figures and Tables ................................................................................................. 6  
Introduction ......................................................................................................................... 7  
  Project requirements ......................................................................................................... 7  
Hardware Design .................................................................................................................. 8  
  Prior Art ............................................................................................................................ 8  
Chassis Design .................................................................................................................... 11  
  Drive System .................................................................................................................... 11  
  Frame ............................................................................................................................... 13  
Electronics ............................................................................................................................ 14  
  Power Management ........................................................................................................ 15  
Control .................................................................................................................................. 17  
  Controller Design ............................................................................................................ 17  
    Linear Translation ......................................................................................................... 17  
    Rotation .......................................................................................................................... 19  
  Controller Implementation ............................................................................................... 20  
    Generation 1 Controller ................................................................................................. 21  
    Generation 2 Controller ................................................................................................. 22  
Future Work ......................................................................................................................... 24  
  Further Controller Development ..................................................................................... 24  
Sensor Integration ............................................................................................................... 25  
Appendix 1: Arduino Code ................................................................................................. 27  
Appendix 2: Java Code ....................................................................................................... 30  
References ............................................................................................................................. 39
List of Figures and Tables

Figures

Figure 1: First iteration of mobile platform .......................................................... 8
Figure 2: Single platform module ........................................................................ 9
Figure 3: Second iteration of mobile platform ..................................................... 10
Figure 4: Graph describing the motor power used in producing motion at a given Angle ........................................................................................................ 12
Figure 5: SyrRen 25 Controller wired into mobile platform .................................. 15
Figure 6: Graphical representation of the motors attached to the chassis .......... 17
Figure 7: Velocity contribution from each wheel projected onto desired velocity vector ........................................................................................................ 18
Figure 8: Graph showing contributions of each motor to velocity while moving in a given direction ................................................................. 19
Figure 9: User Interface of controller application .............................................. 20
Figure 10: Generation 3 controller ....................................................................... 25

Tables

Table 1: Summary of functional requirements, in no particular order ................. 7
Table 2: Angles and motor velocities corresponding to button presses .............. 22
Introduction

Lichtballett, or light ballet, is the brainchild of multimedia and technology based artist Otto Piene. Otto came to fame after forming Group Zero with fellow artist Heinz Mack, a group which began a popular movement of contemporary art in Europe.

One of Otto's most famous set of works is the light ballet. The ballets started off as lamps on hand-cranked turntables set inside of perforated stencils. The light would shine though the perforations in the stencils and project patterns onto the walls surrounding the piece. As Otto began to mechanize his pieces they grew in complexity, incorporating numerous lights, switches, and rotating fixtures that would play with the light as it was being displayed. Otto describes the effect as "the steady flow of unfurling and dimming, reappearing and vanishing light."

The goal of this project is to transform the light ballet from a stationary exhibit to a mobile, interactive experience.

Project requirements

Light ballet is more of an artistic challenge than an engineering one. Maintaining the feel of a light ballet performance is the first and foremost concern of the project's sponsor; the esthetics of the platform itself take a back seat to its function. That being said, the design should also not distract from the performance of light. A summary of functional requirements is listed in Table 1.

<table>
<thead>
<tr>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>System must be able to reliably and continuously perform for six hours.</td>
</tr>
<tr>
<td>System must be capable of handling three, separate, half-hour performances</td>
</tr>
<tr>
<td>with approximately two hours in between each performance.</td>
</tr>
<tr>
<td>System must have smooth and continuous movement, with minimal visible</td>
</tr>
<tr>
<td>vibration.</td>
</tr>
<tr>
<td>System must be safe to operate around children.</td>
</tr>
<tr>
<td>System must be simple to maintain with little attention from museum staff.</td>
</tr>
<tr>
<td>System may be controlled by an operator.</td>
</tr>
<tr>
<td>System must be manufacturable and repeatable.</td>
</tr>
<tr>
<td>System must have a robust controller that will not fail within the two</td>
</tr>
<tr>
<td>month exhibition period.</td>
</tr>
</tbody>
</table>

Table 1: Summary of functional requirements, in no particular order.
Hardware Design

This project had two main components, the design of a robotic platform, which would be replicated three times to create a new fleet of mobile light ballets. The other was the design of the control system. I will discuss both tasks in detail.

Prior Art

This project was worked on before by Steven Jorgensen and Bee Vang as a UROP project, thus a lot of the hardware design had been iterated enough to provide a clear direction of improvement.

The first design was a highly modular square platform that used four omni-directional wheels mounted 90 degrees from each other around the platform, as shown in Figure 1.

Figure 1: First iteration of mobile platform. a) CAD model of the frame; b) frame after assembly. Note that this platform does not include a fixture to mount the light ballet.

The beauty of this design was that because each of the four sides of the platform were identical, only one module needed to be designed. Once the module, seen in Figure 2, was completed, the module could be fabricated four times and joined together to create the platform.
This design had the benefit of being very simple to manufacture, however there were still several issues to be resolved. The first was there was no fixture designed into the frame on which to mount the light ballet. The second was a clearance issue, the platform was too close to the floor, meaning that when the platform traveled over uneven terrain it was prone to getting stuck. The third was a traction problem. Without suspension, having four points of contact with the floor over-defines the system. That means that when set on a surface that is perfectly flat, one wheel most likely will not be in contact with the ground. This makes slipping an issue, which makes the chassis difficult to accurately control.

The second iteration of the chassis switched to a three wheel drive system, which fixes the issues that arose in the four wheel solution without having to add in a suspension system. This comes at the cost of creating a less modular assembly, however significant cost is saved by removing a wheel-motor assembly from the chassis and reliability is enhanced.
Figure 3: Second iteration of mobile platform. Frame is made of 1” square, 1/8” thick steel bar stock, welded together at the joints. This iteration includes a mounting fixture for the light ballet.

This design solved the major issues with the old chassis, but introduced new ones that were mainly assembly based. For one, the motors were too tall for the frame, so in order to mount the motors into the chassis the steel bars had to be cut out post-assembly. The frame is also extremely heavy, which reduces battery life and maneuverability. This can be reduced by changing materials and reducing the assembly’s complexity wherever possible.
Chassis Design

The present iteration of the mobile platform was designed to address concerns laid out in the prior art section of this paper, mainly the issues with fitting the motors within the chassis and weight reduction. The chassis design was completely re-evaluated and deconstructed to ensure that each component was included in the design for a reason, and that assembly would be as simple as possible. That was achieved by limiting the variety of stock chosen to work with, creating symmetry within the system, and reducing redundancy in the design.

Drive System

There were several different drive systems that could have been chosen to power the mobile light ballet, however after much consideration a three omni-directional wheel configuration was selected.

Having only three wheels contacting the floor removes the need to add suspension into the design, as it only takes three points of contact to define a plane. This means that no matter how uneven the surface, as long as the platform has clearance it will drive will each wheel in full contact with the floor. This is an important fact to consider; the final design will be replicated three times, so avoiding additional complexity and cost becomes three times more important.

Using this drive system also allows the platform to travel in any direction and in any orientation. The functional requirements for the project, listed in Table 1, state that the platform must be capable of moving smoothly and continuously through a space. Having the ability to move in any direction and in any orientation allows that goal to be accomplished, where using a two wheel drive system would have struggled.
**Wheels**

The wheels selected were eight inch diameter dualie omni-wheels, sourced from the well-known First Robotics part supplier AndyMark®. These wheels have two features that made them especially attractive for this application. For one, each wheel is actually a system of two wheels which are attached together. This means that the wheels can have a constant diameter while rotating, without compromising the discretization of rollers, which is a common problem with single omni-wheel solutions.

The second is the wheel’s relatively large diameter, which allows for significant clearance between the bottom of the platform and the floor. A problem the previous iteration platform continually encountered was stalling on uneven terrain. A four inch radius will easily allow the platform to have two inches of clearance, which should eliminate this problem.

**Motors**

As the project requirements state that motion must be smooth and continuous, the selected motors have to be powerful enough to move the chassis without stalling. Using equations of motion documented in the Control section of this paper, analysis was done to determine power requirements for the system.

![Graph describing the motor torque used in producing motion at a given Angle. The orange line shows the total percentage of the source voltage used by the platform’s motors while moving in any direction. This graph shows supply voltage is not linear, and ranges between 58 and 67 percent of the supply’s voltage.](image)

**Figure 5:** Graph describing the motor torque used in producing motion at a given Angle. The orange line shows the total percentage of the source voltage used by the platform’s motors while moving in any direction. This graph shows supply voltage is not linear, and ranges between 58 and 67 percent of the supply’s voltage.

Figure 5 shows the percentage of each motor’s total power contributes to moving the platform given a desired angle. Calculations show that the platform will never use 100% of its available motor power while engaging in pure translation. The total power varies between 58% and 67% of available
power. To design around this, a combination of three of whatever motor is selected must be able to drive the platform while performing at half capacity.

The Society of Robots provides a calculator for selecting motors for robotic applications. They do this by calculating a value called the Robot Motor Factor (RMF)\textsuperscript{iii}, which takes into account the desired velocity and acceleration of the robot, as well as its weight and wheel diameter. The RMF of the robot must be less-than or equal to the total RMF of the driving motors which is calculated as the motor’s stall torque multiplied by its no load speed.

For this application, the robot’s RMF is 140, assuming a slight incline of ten degrees. The NPC-41250 geared motor\textsuperscript{iv} with gear ratio of 48.5:1 was selected as a powerful and easy to interface option. According to documentation, stall torque is 260 lbf-in, and the no load speed is 93 RPM, which gives a RMF of 403, which is more than powerful enough for this application. The 90 degree gearbox also lends itself nicely to mounting in a compact package.

Frame

The frame design was kept fairly consistent with the second iteration frame. The revised frame is a bit wider to accommodate larger batteries, while also allowing the motors to be mounted horizontally. Instead of creating complicated battery fixtures as in the prior design, trays were created to hold the battery in place, while still allowing for easy removal from the top of the frame.

![Figure 6: Current platform chassis. This frame is made from 1/8" thick aluminum bar stock. a) frame post-welding, partially assembled, b) cad model of completed frame, including peripherals such as motors, wheels, and batteries.](image)

The fixture was also bared down to three pegs welded onto the frame. This will allow for a variety of different ballet modules to be interfaced with the platform, as long as they have matching female pegs to interface with the chassis.
Electronics

The electronics for the mobile platform were chosen from the top down. Due to the requirements described by the project’s sponsors, for ease of control and upgradability, I decided to develop a purely software solution as opposed to developing a hardware based controller. This provides a much greater potential for future development of software, including setting up routines and path planning, however the scope of this thesis only covers a basic, joystick-like controller which independently dictates translation and rotation of the platform.

A software-based solution was developed on the Android platform, mainly due to its developer friendly architecture and the author’s familiarity with Java. A tether between the controller and the platform would be both unreasonable and not fulfill program requirements, so a wireless solution was needed. Bluetooth was selected as the wireless communication platform, as Bluetooth devices are largely immune to RF noise from other infrastructures, such as WiFi and other Bluetooth devices, and have a long range coupled with low power requirements.

Arduino Uno

The Arduino platform was selected for its rapid prototyping capability and modularity. Communication with other devices through a serial link is also simple to implement. Due to the open source nature of the technology, there are many libraries to support more complex functions for future expansion. An idea that will be implemented outside of this thesis will be the integration of sensors to detect obstacles, and an inertial measurement unit (IMU) to return position feedback to the controller.

Bluetooth Module

The Bluetooth module selected was the BlueSMiRF Silver Bluetooth modem, supplied by sparkfun. The BlueSMiRF modem is a class 2 Bluetooth module, which is the mid-range power option of the two standard platforms. Class 2 modems have a communication range of approximately 20 meters, which is more than enough for this application. The connection is encrypted, which provides additional security, and communicates serially, which makes interfacing with the Arduino platform simple.

Motor Controller

The motor controllers selected for this application is the SyRen 25 motor driver, developed by Dimension Engineering. These motor controllers were selected for their ease to interface with, high amperage capacity, high switching frequency, and versatility. As for the interface with other electronics, all the SyRen 25 needs to operate is a connection to a power source, connection to a motor, and control signal from an external source.
The SyRen has several operating modes for controlling the attached motor. The selected operating mode uses a single voltage signal from an outside source to command its motor, which ranges from zero to five volts. This signal can be provided by any source, however the architecture lends itself nicely to the Arduino’s servo library. The servo library contains the write() function, which outputs a signal from zero to five volts using pulse width modulation (PWM). A 5V signal (a PWM value of 180 using the write() function) corresponds to full forward, 0V (0) corresponds to full reverse, and 2.5V (90) corresponds to full stop. This operating mode greatly simplified testing the system, as the controller could be attached to a potentiometer splitting a 5V signal to validate functionality.

Power Management

The project requirements state that the entire light ballet installation should be able to continuously perform basic functions, such as slow platform rotation and operation of the light and turntable system, for a minimum of six hours. The system should also be capable of full operation for at least thirty minutes without needing to charge. By experimentation it was determined that the best way to achieve this goal would be to use two lead-acid 12 volt, 55 amp-hour (Ah) batteries and a 220 to 12 volt converter in parallel to provide power for the entire system. The converter can be plugged directly into a power source using a retractable tether run along the ground. The tether would allow the light ballet to run indefinitely with limited chorography. Removing the tether then allows for a full range of motion.
Two separate battery life tests were performed on the moving platform and light turntables. The moving platform was run continuously at half of its maximum speed (a value of 135), on a pair of 12V, 18Ah batteries, which total to 36 Ah when connected in parallel. Motion persisted without degrading for 3 hours and 47 minutes. A similar test was performed on the light fixture, which lasted for 57 minutes before the light noticeably began to dim. These two tests suggest that if the two systems were run in parallel, two 55 Ah, 12V batteries should be able to power both systems for half an hour, with a safety factor of about 6.
Control

As discussed in the previous section, control of the mobile platform was decoupled into two compartmentalized portions. One is translation, where the platform chassis moves in the x-y direction across the floor without changing its orientation. The other is rotational, where the robot chassis will change its orientation in place, without moving in the x-y plane. Using simple math, these two types of motion can be combined into a coherent motion plan for the mobile platform.

Controller Design

The platform is designed with three omni-directional wheels attached to the frame, each mounted exactly 120 degrees from each other. A representation is shown in Figure 8 that shows the wheel’s orientation compared to the chassis, and the vector contribution to the chassis’ velocity when the motor is run at full speed forward. Assuming that the front of the chassis lies on the y-axis, motors 1, 2, and 3 are mounted at 60, 180, and 300 degrees respectively.

Linear Translation

Calculating linear equations of motion is a matter of projecting vectors from the wheels onto the desired velocity vector at the chassis center, demonstrated in Figure 9.
Figure 9: Velocity contribution from each wheel projected onto desired velocity vector. θ represents desired direction of travel, and the length of the red vector describes the desired speed. The desired velocity vector, shown in red, is centered on the chassis.

The velocity of each wheel would normally be calculated by projecting the desired velocity vector onto the vector described by each wheel’s forward path of travel, as described by Equation (1).

\[ V = v \cdot \cos (\phi - \theta) \]  

Equation (1)

Here \( v \) is the desired velocity, \( \phi \) is the forward drive direction of the wheel in question, and \( \theta \) is the desired direction of travel. However, because the forward drive direction of the wheel is shifted 90 degrees from the mount direction, cosine is switched to sine to account for the offset. This results in Equations (2) through (4), which describe the resultant velocity of the three wheels when given a desired velocity magnitude, \( v \), and direction, \( \theta \).

\[ V_1 = v \cdot \sin(60 - \theta) \]  

Equation (2)

\[ V_2 = v \cdot \sin(180 - \theta) \]  

Equation (3)

\[ V_3 = v \cdot \sin(300 - \theta) \]  

Equation (4)

From these three equations, we can see that when the platform is traveling in the direction that one of the wheels in mounted in its contribution will be reduced to zero, and when traveling...
perpendicularly to the wheel mount direction the wheel’s contribution will be maximal, which matches intuition.

![Graph showing contributions of each motor to velocity while moving in a given direction.](image)

**Figure 10:** Graph showing contributions of each motor to velocity while moving in a given direction.

**Rotation**

Controlling rotation is a much simpler problem. The wheels are mounted perpendicular to the platform’s axis of rotation, so to rotate the chassis each motor simply has to be given an identical voltage that is either higher, or lower than the neutral voltage of 2.5V. The angular velocity of the robot will then be the speed of the wheels, divided by the distance the wheel is mounted from the center of the chassis. Positive values will cause the platform to rotate to the right, and negative values will cause it to rotate to the left.

Combining angular rotation and linear translation into one master equation allows for a complete control of the platforms motion in 2D space. This combination is described in Equations (5) through (7) below.

\[
V_1 = v \cdot \sin(60 - \theta) + \gamma \quad (5)
\]

\[
V_2 = v \cdot \sin(180 - \theta) + \gamma \quad (6)
\]

\[
V_3 = v \cdot \sin(300 - \theta) + \gamma \quad (7)
\]
Controller Implementation

The controller was designed as an Android application that could be posted into any Bluetooth capable Android device to control the robotic platforms. The final goal is to be able to control three platforms with a single application, however here I will only discuss a single platform controller. It is assumed that to scale up the controller, the written code would only need minor alterations.

The platform was designed modularly in a master-slave communications protocol between the Android device and Arduino Uno. The Android device handles interaction with the user, and interprets the user input into coherent packets which can be sent over Bluetooth. The Arduino Uno receives and interprets these packets and translates them into motion of the robotic platform.

Figure 11: User Interface of controller application. a) Generation 1 controller. Four buttons control motion in cardinal directions, as well as a stop button. Volume buttons control left and right rotation. b) Generation 2 controller. Central widget is a joystick-like apparatus which logs user touch interactions to control platform translation. Slider at the bottom of the screen controls platform rotation.
Generation 1 Controller

The generation 1 controller was designed mainly as a test of Bluetooth communication. It validated the concept that long strings of data could be communicated using Bluetooth technology between the Android platform and the Bluetooth module tied to the Arduino. Once the Bluetooth communication concept was validated, the collected data was used for validating further experiments with other controllers.

Java Code: Master

The generation 1 controller used five buttons to control the motion of the platform, a button for each cardinal direction (north, south, east and west, or in this case forward, reverse, left and right). These buttons could control translation of the platform in only these four directions, which greatly limited the scope of what the controller could do. This functionality was improved somewhat by adding in rotation functionality, which was achieved by pressing the phone’s “Volume Up” and “Volume Down” buttons.

The master code was designed as modularly as possible, to leave room for future expansion. Button presses altered global variables, magnitude, and angle, where magnitude was a preset value less than 90 to keep motors from saturating, and angle was altered based on which button was pressed.

To calculate the command signal to each motor, the magnitude and angle set by pressing a button are passed into a helper function. The function uses Equations (5) through (7) to calculate the “speed” of each motor, using magnitude as \( v \) and angle as \( \theta \). The neutral command signal is then added to the calculated “speed” to generate the final command signal, which will vary between 0 and 180. There are, of course, other conversions and calculations involved; for more detail, see
Appendix 2: Java Code.

The ideal results for pressing a button are summarized in Table 2. These values were used extensively for testing and debugging and validating the Generation 2 controller.

<table>
<thead>
<tr>
<th>Button</th>
<th>Angle</th>
<th>Motor 1 Command Signal</th>
<th>Motor 2 Command Signal</th>
<th>Motor 3 Command Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>0</td>
<td>109</td>
<td>90</td>
<td>70</td>
</tr>
<tr>
<td>Reverse</td>
<td>180</td>
<td>70</td>
<td>90</td>
<td>109</td>
</tr>
<tr>
<td>Left</td>
<td>270</td>
<td>78</td>
<td>113</td>
<td>78</td>
</tr>
<tr>
<td>Right</td>
<td>90</td>
<td>101</td>
<td>67</td>
<td>101</td>
</tr>
</tbody>
</table>

Table 2: Angles and motor velocities corresponding to button presses. Magnitude is a constant set to 40. Command signal is the velocity of the motor plus a constant of 90, which represents the neutral state of the motor.

The command signals were then sent to a final helper function to handle the Bluetooth communication with the Arduino Uno. This function took the command signals and bundled them into a command packet to send over serial. The benefit of serial communication is that information can be passed as ascii characters, which means that a simple String can be used to pass data across the Bluetooth channel. The general form of the command packet is shown in Equation (8).

\[M1###M2###M3###!\]

(8)

The “###” following the “M1”, “M2”, and “M3” represents the command signal to be sent to the corresponding motor, which can be a number anywhere from 0 to 180. The “!” at the end is a marker letting the slave know when the command packet has finished transmitting.

**Arduino Code: Slave**

The Arduino’s only job is to continually wait for a signal from its master program. Upon receiving serial data, it reads the data until encountering the stop character, and sends that packet to a helper function be interpreted. The packet is then separated by the “M1”, “M2”, and “M3” delimiters and the command is sent to each of the motors.

If the Arduino reads data in that doesn’t make sense, such as in the case where data is corrupted or the Bluetooth connection fails, the motors automatically stop running.

**Generation 2 Controller**

The generation two controller, seen in Figure 11: User Interface of controller application. a) Generation 1 controller. Four buttons control motion in cardinal directions, as well as a stop button. Volume buttons control left and right rotation. b) Generation 2 controller. Central widget is a joystick-
like apparatus which logs user touch interactions to control platform translation. Slider at the bottom of
the screen controls platform rotation.-b works in the way a user would most likely expect to control a
mobile robotic platform that is capable in moving in any direction. The controller features a joystick-like
widget in the center of the screen that responds to user touch by moving the purple controller head
beneath the user’s finger. The slider at the bottom of the screen determines the rotation of the platform.
While centered, the platform has no additional rotation. When slid to the right, the platform rotates to
the right, and when slid to the left, the platform rotates to the left.

Java Code: Master
The generation two controller went through several design iterations to get to its final state as
shown above. The first attempt had the master program compute the motor speeds and send packets
like in the generation one controller. The program attempted to send a data packet every time it
detected a change in the user’s finger position. However, it was discovered that it is possible to flood the
Bluetooth connection with too much data, which causes the chassis to perform erratically.

To remedy this, the master-slave protocol was retooled. Instead of calculating the motor
velocities on the master-side, the motor control calculations were outsourced to the Arduino slave. This
allows the master to send only magnitude and direction information to the Arduino. This simplified the
outgoing packet to a maximum of nine characters, which is a drastic savings from the original sixteen.
The simplified packet is described in Equation (9), where nomenclature is the same as Equation (8).

\[ M###A###! \]

(9)

To address rotation, the slider was added in. The slider is paired with a listener that only
updates when the user changes the slider’s position. Because of the Arduino slave’s listening
architecture, commands can be separated into smaller pieces for transmission. Mainly this means that
each packet does not need to contain a magnitude, angle, and rotation command for the interpreter to
correctly handle the data. Taking advantage of this, the rotation listener individually sends a rotation
packet only upon alteration. Because motor speed saturates at 90, rotation can only range from 0 to 90,
which limits the rotation command packet to a maximum of four characters. This rotation packet is
described in Equation (10).

\[ R##! \]

(10)

To further reduce the amount of data being sent at once, code was altered to only send an
update to the Arduino slave when the value has been updated. This occasionally reduces the size of the
translation packet from Equation (8) to half of its size.

Arduino Code: Slave
The Arduino slave’s main listening operations are largely the same as the Generation 1
controller, which is one of the many benefits of the master-slave communication architecture. The
delimiters where changed to “M”, “A”, and “R”, which stand for magnitude, angle, and rotation
respectively. By using Equations (5) through (7) the motor command signals are generated and sent to the motors. The same safety features are implemented.

Future Work

This project is being used as an actual art installation in Berlin starting in August 2014, and as such work on improving both the physical design and controller will continue indefinitely.

Further Controller Development

Now that fine control is proven to work with the generation two controller, the third generation controller is now under development. This next iteration of controller will allow the user to drag the robot icon along a desired path for rudimentary path planning. As of now, there is no position feedback implemented in hardware, so knowing exact position will be impossible. The proposed method for making this controller function will be to use open loop control and timers to roughly estimate relative position.
Figure 12: Proposed generation 3 controller. Users can press the purple dot representing the robot and drag it around the screen for rudimentary path planning. The black segment of line represents the portion of the path the application believes the platform has already traversed, while the grey segment shows the future path. The arrow displays the calculated orientation of the robot.

Future generations of controllers will implement different features. The generation four controller is planned to have basic functionalities preprogrammed, which will be selectable by a dropdown menu and button press. These functionalities will handle the six-hour, unmanned showcases that are scheduled to happen every night for four months. The final iteration will replicate whatever level of control was achieved for one platform for the simultaneous control of three platforms.

**Sensor Integration**

Sensors can be added into the mobile platform to improve performance. The addition of an ultrasonic distance sensor array and IMU allows for greater control accuracy and safety.
An array of at least six ultrasonic distance sensors can be placed on the perimeter of the chassis to detect obstacles in the path of the platform mid motion. If an object is detected by the ultrasonic distance sensor the platform will quickly slow to a stop, then reverse in the opposite direction of the obstacle. This will ensure the safety of museum patrons while wandering through the exhibit space, while also passively preventing the three platforms from crashing into each other mid choreography.

An IMU will enable position and orientation feedback. A fairly inexpensive analog IMU includes a gyroscope, magnetometer, and accelerometer. With these three instruments, the position and orientation of the platform can be inferred using the starting point as a reference, and the orientation of the platform can be checked against the earth’s magnetic field for recalibration. Integrating the IMU will replace timers in approximating position, at a cost of processing power. If necessary, a more powerful microcontroller can be used to handle the load, such as Maple board produced by Leaf Labs viii.
Appendix 1: Arduino Code

#include <Servo.h>

// Motor Pins
#define MOTOR1 3 // 0 Degrees
#define MOTOR2 5 // 120 Degrees
#define MOTOR3 6 // 240 Degrees

#define NEUTRAL 90
#define RADIUS 140.0
#define MAXSPEED 40

double SPEED_CONST = MAXSPEED/RADIUS; // int equals max speed of robot

Servo m1,m2,m3;

int s1,s2,s3;
int ang, mag, rot = 0;

void setup(){
    Serial.begin(115200);
    m1.attach(MOTOR1);
    m2.attach(MOTOR2);
    m3.attach(MOTOR3);
    delay(15);
    stopMotors();
}

void loop(){
    readData();
    setMotors();
}

// Send command signals to the motors, mag, ang, and rot.
void setMotors(){
    s1 = NEUTRAL + mag * SPEED_CONST * sin(convertToRadians(180-ang)) + rot;
    s2 = NEUTRAL + mag * SPEED_CONST * sin(convertToRadians(60-ang)) + rot;
    s3 = NEUTRAL + mag * SPEED_CONST * sin(convertToRadians(300-ang)) + rot;
    m1.write(s1);
    m2.write(s2);
    m3.write(s3);

    // Serial.print("M1: ");
    // Serial.print(s1);
    // Serial.print(" M2: ");
    // Serial.print(s2);
    // Serial.print(" M3: ");
    // Serial.println(s3);
}

// Send command to stop motors from moving
void stopMotors(){

```java
mag = 0;
ang = 0;
setMotors();
}

//Converts angle in degrees to radians
double convertToRadians(int num)
{
    double angle = PI*(double)num/180;
    return angle;
}

// blocking serial reader
String serialRead()
{
    char in;
    while (in = Serial.read() == -1) {
    }
    return String(in);
}

// Reads in one packet of data from Bluetooth receiver
void readData()
{
    //Serial.println("Read data entered!");
    String temp = "";
    String val = serialRead();
    while (val != ","){
        temp += val;
        val = serialRead();
    }
    handlePacket(temp);
    //Serial.println("Read data exited!");
}

// handles packet data, setting global variables accordingly
void handlePacket(String c){
    char charBuf[4];
    int holdValue;
    while (c.length() != 0){
        // looks for the first not number character and records the index in holdValue
        for(int i=1;i<5;i++){
            if(c[i+1]-'0'<=0 || c[i+1]-'0'>9){
                holdValue = i-1;
            }
        }
        if (c[0] == 'M'){
            mag = c.substring(1,2+holdValue).toInt();
            // Serial.print("Magnitude: ");
            // Serial.println(mag);
        } else if (c[0] == 'A'){
            ang = c.substring(1,2+holdValue).toInt();
        }
    }
}
```
Serial.print("Angle: ");
Serial.println(ang);

else if (c[0]=='R') {
    rot = c.substring(1,2+holdValue).toInt();
    Serial.print("Rotation: ");
    Serial.println(rot);
}
else{
    stopMotors();
    c=c.substring(1);
}
  c = c.substring(2+holdValue);
Appendix 2: Java Code

activity_main.xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <com.example.joysticktester.CustomSurfaceView
        android:id="@+id/CSV"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_margin="10dp"/>

    <SeekBar
        android:id="@+id/rotation_bar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/CSV"
        android:layout_alignLeft="@+id/CSV"
        android:layout_margin="10dp"
        android:max="100"
        android:progress="50"
        android:secondaryProgress="0"/>

</RelativeLayout>
package com.example.joysticktester;

import android.content.Context;
import android.graphics.*;
import android.os.AsyncTask;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;

public class CustomSurfaceView extends SurfaceView implements SurfaceHolder.Callback {

    private MySurfaceThread thread;
    private Paint paint;
    private double x, y, dx, dy, angle, magnitude;
    private boolean run = true;
    private Bitmap joystick_ball, limit_ring;
    private int zx, zy, print_angle, print_magnitude;

    private final double RADIUS = 140;

    public synchronized int getAngle() {
        return print_angle;
    }

    public int getMag() {
        return print_magnitude;
    }

    public CustomSurfaceView(Context context) {
        super(context);
        init();
    }

    public CustomSurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    public CustomSurfaceView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        init();
    }

    private void init() {
        thread = new MySurfaceThread(getHolder(), this);
        getHolder().addCallback(this);
        paint = new Paint();
        paint.setTextSize(40);
        paint.setColor(Color.rgb(255, 0, 0));
        joystick_ball = BitmapFactory.decodeResource(getResources(),
                R.drawable.joystick_ball);
    }

    // Other methods and logic...
}
limit_ring = BitmapFactory.decodeResource(getResources(), R.drawable.limit_ring);
    zx = this.getWidth()/2;
    zy = this.getHeight()/2;
    x = zx;
    y = zy;
}

@Override
public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2, int arg3) {

}

@Override
public void surfaceCreated(SurfaceHolder arg0) {
    thread.execute((Void[])null);
}

@Override
public void surfaceDestroyed(SurfaceHolder arg0) {
}

protected synchronized void onDraw(Canvas canvas){
    super.onDraw(canvas);

    zx = canvas.getWidth()/2;
    zy = canvas.getHeight()/2;

    canvas.drawRGB(255,255,255);
    canvas.drawText(Double.toString(dx),20,50,paint1);
    canvas.drawText(Double.toString(dy),20,100,paint1);
    canvas.drawText(Double.toString(print_angle),500,50,paint1);
    canvas.drawText(Double.toString(print_magnitude), 500, 100, paint1);
    canvas.drawBitmap(limit_ring, (canvas.getWidth()-limit_ring.getWidth())/2, (canvas.getHeight()-limit_ring.getHeight())/2, paint1);

    if(x == 0 && y == 0){
        canvas.drawBitmap(joystick_ball, (canvas.getWidth()-
            joystick_ball.getWidth())/2, (canvas.getHeight()-
                joystick_ball.getHeight())/2, paint1);
    } else {
        canvas.drawBitmap(joystick_ball, (float) x-
            joystick_ball.getWidth()/2, (float) y-joystick_ball.getHeight()/2, paint1);
    }
}

public class MySurfaceThread extends AsyncTask<Void, Void, Void>{
    SurfaceHolder nSurfaceHolder;
    CustomSurfaceView cSurfaceView;

    public MySurfaceThread(SurfaceHolder sh, CustomSurfaceView cv) {
nSurfaceHolder = sh;
cSurfaceView = csv;

cSurfaceView.setOnTouchListener(new OnTouchListener{

    @Override
    public boolean onTouch(View v, MotionEvent e) {
        // TODO Auto-generated method stub
        x = e.getX();
        y = e.getY();
        calculateValues(x, y);

        switch(e.getAction() & MotionEvent.ACTION_MASK){
            case MotionEvent.ACTION_DOWN:
                break;
            case MotionEvent.ACTION_UP:
                x=zx;
                y=zy;
                dx=dy=0;
                print_magnitude = 0;
                break;
            case MotionEvent.ACTION_CANCEL:
                print_magnitude = 0;
                break;
            case MotionEvent.ACTION_MOVE :
                break;
        }
        return true;
    }

    private synchronized void calculateValues(double xx, double yy) {
        // TODO Auto-generated method stub
        dx = (xx-zx);
        dy = (yy-zy);
        angle = Math.atan2(dy, dx);
        magnitude = Math.sqrt(dx*dx+dy*dy);
        int pre_print_angle = (int)(angle*180/Math.PI)+90;
        if (magnitude > RADIUS){
            x = (RADIUS*Math.cos(angle)) + zx;
            y = (RADIUS*Math.sin(angle)) + zy;
        }
        dx = (x-zx);
        dy = (y-zy);
        print_magnitude = (int)Math.sqrt(dx*dx+dy*dy);
        if (pre_print_angle < 0){
            print_angle = pre_print_angle+360;
        } else print_angle = pre_print_angle;
    }

};
@Override
protected Void doInBackground(Void... arg0) {
    // TODO Auto-generated method stub
    while(run){
        Canvas canvas = null;
        try{
            canvas = nSurfaceHolder.lockCanvas(null);
            synchronized(nSurfaceHolder){
                cSurfaceView.onDraw(canvas);
            }
            Thread.sleep(10);
        }
        catch(InterruptedException e){
        }
        finally{
            if(canvas != null){
                nSurfaceHolder.unlockCanvasAndPost(canvas);
            }
        }
    }
    return null;
}
MainActivity.java

package com.example.joysticktester;

import java.io.IOException;
import java.io.OutputStream;
import java.lang.reflect.Method;
import java.util.UUID;
import android.os.Build;
import android.os.Bundle;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.util.Log;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.Toast;

public class MainActivity extends Activity {
    private static final String TAG = "JoystickTester";

    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private OutputStream outStream = null;

    private CustomSurfaceView csv;
    private Thread thread;

    private String packet;// lastPacket;

    private int rotation;
    private SeekBar rotationControl = null;

    // SPP UUID service
    private static final UUID MYUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    // MAC-address of Bluetooth module (you must edit this line)
    private static String address = "00:06:66:08:10:99";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        thread = new BluetoothManager();
        setContentView(R.layout.activity_main);
        btAdapter = BluetoothAdapter.getDefaultAdapter();
        csv = (CustomSurfaceView) findViewById(R.id.CSV);
        checkBTState();
        rotationControl = (SeekBar) findViewById(R.id.rotation_bar);
        rotationControl.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
            {35
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    rotation = (int)((progress - 50) / 2.0);
    sendData("R" + rotation + "!");
}

public void onStartTrackingTouch(SeekBar seekBar) {
}

@Override
public void onStopTrackingTouch(SeekBar arg0) {
}

private BluetoothSocket createBluetoothSocket(BluetoothDevice device) throws IOException {
    if (Build.VERSION.SDK_INT >= 10) {
        try {
            final Method m =
            device.getClass().getMethod("createInsecureRfcommSocketToServiceRecord", new Class[] {
                UUID.class });
            return (BluetoothSocket) m.invoke(device, MY_UUID);
        } catch (Exception e) {
            Log.e(TAG, "Could not create Insecure RFComm Connection", e);
        }
    }
    return device.createRfcommSocketToServiceRecord(MY_UUID);
}

@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "...onResume - try connect...");
    // Set up a pointer to the remote node using it's address.
    BluetoothDevice device = btAdapter.getRemoteDevice(address);
    // Two things are needed to make a connection:
    // A MAC address, which we got above.
    // A Service ID or UUID. In this case we are using the UUID for SPP.
    try {
        btSocket = createBluetoothSocket(device);
    } catch (IOException e1) {
        errorExit("Fatal Error", "In onResume() and socket create failed: "+ e1.getMessage() + ".");
    }
    // Discovery is resource intensive. Make sure it isn't going on
    // when you attempt to connect and pass your message.
    btAdapter.cancelDiscovery();
}
if(btAdapter==null) {
    errorExit("Fatal Error", "Bluetooth not support");
} else {
    if (btAdapter.isEnabled()) {
        Log.d(TAG, "...Bluetooth ON...");
    } else {
        //Prompt user to turn on Bluetooth
        Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableBtIntent, 1);
    }
}

private void errorExit(String title, String message){
    Toast.makeText(getBaseContext(), title + " - " + message, Toast.LENGTH_LONG).show();
    finish();
}

private void sendData(String message) {
    byte[] msgBuffer = message.getBytes();
    Log.d(TAG, "...Send data: " + message + "...");
    try {
        outStream.write(msgBuffer);
    } catch (IOException e) {
        String msg = "In onResume() and an exception occurred during
write: " + e.getMessage();
        if (address.equals("00:00:00:00:00:00"))
            msg = msg + "\nUpdate your server address from
00:00:00:00:00:00 to the correct address on line 35 in the java code";
        msg = msg + "\nCheck that the SPP UUID: " +
MY_UUID.toString() + " exists on server.\n"
        errorExit("Fatal Error", msg);
    }
    //}
}

private void commandMotors(){
    int angle = csv.getAngle();
    int mag = csv.getMag();
    int a = csv.getAngle();
    int m = csv.getMag();
    while (angle == a && mag == m){
        a = csv.getAngle();
        m = csv.getMag();
        Log.d(TAG, "Mag: "+m+" Angle: "+a);
    }
    packet = "";
if (m!= mag){packet += "M" + m;}
if (a!= angle){packet += "A" + a;}
packet += ";
Log.d(TAG, packet);
sendData(packet);
}

public class BluetoothManager extends Thread {
    public void run(){
        while (true){
            try {
                commandMotors();
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
References

[1] listart.mit.edu/node/693