Technology and the Stage:

Achieving Control Through The Kinect/Arduino Interface

By Jeff Hammel and Matthew Parmelee

Introduction

The recent explosion of interest in open-source microcontrollers from hobbyists and programmers alike over the past decade is a clear indicator of growing expansion in the capabilities of such a simple platform. Realizing this, and having some experience in microcontrollers and robotics between the two of us, we made it our goal to demonstrate the limitless real-world applications of this hardware.

We set out to create a prototype proof-of-concept robotics platform for Dr. Chang. The robot is controlled by the Xbox Kinect controller and is able to respond to movements and programmer predefined motion gestures made by an actor onstage. Given the scope of Dr. Chang's project, the Kinect programming and software was handled by Bin Gao, Yang Hu, and QiZhang Dai, while our team handled the robotics and microcontroller platforms. Our robot consisted of an Arduino Uno microcontroller board and a RobotBits robot shield attachment for the Arduino. The robot was assembled by Jeff Hammel and the Arduino programming was written by Matt Parmelee.

The driving motivation behind this project was, simply put, the virtually limitless applications of such a technology. Dr. Chang was looking for a conceptual realization of interaction between the physical world and the virtual world in a theatrical production but the idea of controlling complex hardware with physical movements has the potential to span all fields of technology, from entertainment to medicine. Realizing this, we set out with a proof-of-concept prototype in mind and the desire to demonstrate our hardware configuration as a viable system of human/technology interaction.

At the end of development, our system utilizes every feature we had planned, in addition to many others that were a result either of happenstance or recognizing opportunities for improvement. We succeeded in developing a robotics platform that can be controlled by user input, whether it is from a keyboard or other specialized hardware. We also succeeded in mapping physical hand gestures into usable commands. We were also able to integrate these two parts of our project to be able to control out robotics platform using the mapped physical hand gestures, the final goal of what we initially set out to do. Lastly, and arguably most importantly, we managed to abstract a lot of the finer details of our programs in the hopes that beginners can approach our design with confidence and allow further iterations

of this technology to continue. Given more time, there are limitless improvements and refinements that could be made. However, our project, as planned, was a resounding success.

Parts List

As a result of our design requirements, our fully functional system requires a wide variety of software packages and physical hardware to function properly. Fortunately, all of these parts are easily accessible:

- **Hardware**
  - One laptop running Windows 7
  - One laptop running OSX 10.6
  - Arduino-based robotics platform
  - Arduino microcontroller
  - RobotBits motor shield
  - 9V wall power adapter
  - USB A/B Serial cable
  - Microsoft Kinect
  - Other miscellaneous robotics parts to build our mobile platform

- **Software**
  - Microsoft Kinect drivers for Windows laptop
  - Arduino IDE and Java Serial Libraries for OSX laptop
  - Arduino-side robotics control program (C++)
  - OSX (client)-side command processing program (Java)
  - Kinect (server)-side interface and command generation program (C++)

The only crucial configuration required for this system is the hardware-specific software packages for the Windows and OSX platforms. However, once these are properly configured, the system should operate on relatively any Arduino-based robotics platform with a similar range of motion. Compatibility between the several different aspects was relatively a non issue because each hardware device communicated with their respective laptops individually. While most of these parts were incidental, we will see below that the crucial elements were the Kinect, Arduino, robotics and the laptops.

The Xbox Kinect

The Kinect was a vital piece of hardware in the development of our project. It was developed by Microsoft to track human movement and interpret it into a stream of data to be read by the computer. It uses a range finder camera and an infrared camera to determine body movement and distance from the camera. It is able to pick up everything from full body movements to simple hand gestures. For our project, we decided upon a set of simple hand and arm gestures that would signify that a command was being issued. Microsoft originally developed the Kinect as a game peripheral, but through open source drivers we were able to connect the Kinect to a computer and receive a raw data stream to interpret from the Kinect. As mentioned above, however, our team did not handle the major aspects of the Kinect programming portion of the project. We would like to thank Bin Gao, Yang Hu, and QiZhang Dai again for their programming work with the Kinect to assist us with our project. They implemented a full body tracking system that kept track of each movement made by the user. When a command gesture was recognized, their program registered that a command was issued, determined which command it was, and designated a pre-determined character value to acknowledge which command was given. This character was broadcasted through the server connection for any waiting client (our robot program) to pick up and use for their own purposes.

The Arduino Microcontroller

The Arduino board is an open-source inexpensive microcontroller. Using the freely-available IDE, a user can upload C++ code to the board via a USB connection. Minimally, the Arduino programs require two methods to run properly. The setup() method runs once and initializes all program variables. The loop() method runs constantly for the duration of power being supplied to the board.

The microcontroller features several benefits that were indispensable to the success of our project. Given the difficulty in obtaining the robotics platform quickly, the $25 microcontroller was the first part we obtained. First, as an open-source platform, the development community is massive, providing us with constant examples and demonstrations. Without these, it is unlikely we would have accomplished such complex tasks as the Java Serial Library on time. Secondly, the modular nature of the board allows great expansion to the feature set of our finished system. The motor shield, for example, allows us to easily interface with the robotics motors and abstracts other functions of the board irrelevant to that end. Furthermore, for example, it would be relatively simple to purchase a radio chip attachment and incorporate it into our code base to achieve an ideal fully-wireless setup of the entire system. With these features in mind, the Arduino was a clear choice of hardware.

The Robotics Platform

The RobotBits control shield made up the rest of our controlling electronic hardware. It had to be assembled and soldered together from several pieces ordered from the robotbits.co.uk website. We had a variety of options available to us, but we choose robotbits as our primary source to acquire our hardware to avoid any compatibility errors between the different hardware parts. The shield connects to the Arduino board directly, connecting through the several ports on the Arduino board. We also had 2 low voltage motors that connected the motor shield interface to the wheels of the tank robot for movement. These motors also required an additional external power supply to be able to run. We included a battery pack within the chassis of the robot to provide this extra power, but decided during testing not to rely on the batter power but rather use a 9V power adapter that connected to the Arduino. The batteries could be uses easily if we were to make our robot operate wirelessly. The RobotBits shield does not require explicit programming; it is only the interface between the Arduino and the motors of the robot.

The Kinect Server Program

As mentioned before, the Kinect programming was handled by our partner groups of Bin Gao, Yang Hu, and QiZhang Dai. They designed a program to recognize certain agreed-upon gestures from the Kinect's raw data and simplify the data into a simple character value. Each gesture was assigned a particular character associated with a command. Their server would then broadcast that character to any and all clients connected to it through the network. By simplifying the immense data being generated by the Kinect to simple character values, our client and Arduino programs could easily interpret the signals being sent by the user and apply the correct function accordingly.

The Arduino Program

Due to our difficulty in quickly acquiring the robotics platform and the ease of obtaining a $25 microcontroller, an Arduino debug program was the first piece of software written. Lacking the actual robot, we began by reading over the white paper included on the robotics website, which gave us an understanding of how (ideally) the platform would function upon its arrival and construction. Realizing this, we sought to develop a program that could output command successes in the form of blinks of its on-board LED. This strategy proved to be indispensable and a great time saver in the development process.

The core methods of the Arduino program, as mentioned above, are the setup() and loop() commands that act as the main() on the Arduino platform. The setup() is relatively straightforward, though a great deal of research went into determining the ideal configuration. For instance, the correct board pins had to be mapped properly and assigned variables so the software could communicate and output to the pins. The motor shield features four pins corresponding to the speed (power consumption)

and direction (HIGH or LOW voltage) of each of the two motors. Once these variables were set, we were free to write functions that control the motion of the platform. The next step was to simply write five functions:

- forward() - robot proceeds forward, continues to do so until new command is issued
- backward() - robot pauses (to prevent motor wear), proceeds backwards until new command is issued
- left() - robot pauses, turns ninety-degrees counterclockwise, and calls forward()
- right() - robot pauses, turns ninety-degrees clockwise, and calls forward()
- stop() - robot ceases all motion until new command is issued

The loop() of the program waits until serial input is received over the attached USB cable and uses a switch statement where each of the five cases correspond to a command, issued via a single character ('f' for forward, 'b' for backward, etc.) which in turn calls the appropriate function. The function parameters of speed and duration are hard-coded to correspond to our particular model, but are easily altered for different platforms.

The Client Program

Since the Kinect and Arduino development were occurring side-by-side, the Kinect integration would have to wait to be implemented last. The next logical step would be to develop our client-side program to accept Kinect commands over a wireless network, handle them appropriately, and output the relevant command out to the robotics platform.

Initially, we intended to use the StandardFirmata package for the Arduino IDE, which allows manual control via a C++ GUI of every individual pin on the microcontroller board. However, it was determined this was far more than was needed, and the reverse-engineering of such a large program to design our own specialized version would have been prohibitively expensive in terms of man hours. With this in mind, we sought to develop our own program from scratch.

The first important decision was that of which programming language in which to write the client. While C++ would seem to be the obvious choice since both the Kinect Server and Arduino programs were written in C++, we realized that not only were the Java Serial Libraries easy to configure on an OSX machine but since we would be implementing a socket connection between the Server and Client, the choice of programming language was largely irrelevant. If the runtime complexity had been considerably larger, this choice would have been reconsidered due to Java's memory constraints. Fortunately, this was not the case.

The Java Client, at its core, is a modified version of the SerialTest program, included in the Arduino documentation. It allows a data connection to be established via USB between a laptop and the Arduino interface. As mentioned earlier, the Arduino program waits for input from the serial connection the client program provides. About 90% of this program involves establishing a serial connection to the Arduino.

The program functions first by initializing the Java Socket to the Kinect Server, so an InputStream may be established between the two programs and commands can be exchanged. After initializing several other variables used for raw input and command flagging, the program executes an infinite while loop that:

- Read raw input from the Kinect Server InputStream
- Cast input to a char
- If the received command is flagged, it means the robot is currently already in that state (for instance, we don't want to overload the robot with a stream of forward() requests). If this is the case, we ignore the command and skip the rest of the loop.
- Use a switch statement to assign the command to an integer
- If that integer is in the command set, we output it to the board
- Sleep for one second (to prevent motor fatigue)

If invalid input somehow occurs, the stop() command is issued to prevent the system from requiring a full reboot in the event of erroneous data. While this at no point occurred during testing, it is important to have these fail-safes implemented in the system.

Software Interaction

The three pieces of software outperformed all expectations, with some added benefits we had not predicted. However, there were initially some compatibility hurdles to overcome.

First and foremost was the issue of connecting the two programs. Initially we had planned to run all the software on a single machine, but the Kinect drivers are not compatible with OSX and the Java Serial Library is difficult to configure for Windows. As such, we needed to implement Sockets as an avenue of communication between the two. Furthermore, by running the robotics platform as a client to the Kinect Server, we have left open the potential for a threaded multi-client server capable of controlling many devices from a single gesture. While not a requirement of our project, a large potential for expansion was a prominent design goal.

Once that issue was resolved, the programs simply had to be reconfigured to agree on a single command set (the Server must output commands the Client understands) before they worked in tandem flawlessly. As an added and unplanned bonus, we learned that the operation of the Java Socket provides a stream of characters, which allows us to chain commands and increase the complexity of the robot's range of motion. This feature was disabled in our final release for the sake of testing, but can be reasonably implemented again for development purposes.

While there may be some subtleties to the software interaction, the configuration of the software abstracts much of the hardware from other hardware. As such, the interaction of the hardware was largely flawless and mostly a matter of filtering out irrelevant streams of data from the Kinect's command generation into a single command.

As a whole, the system functions to expectations with minimal quirks. As a result of our flagging algorithm, it is impossible, for example, to make two consecutive right turns as the second would be flagged as a repeated command. However, a command history list would be simple to implement and quickly prevent this from occurring.

Response time on some commands was a limitation not of our software, but rather the detection algorithm used by the Kinect hardware. While this is largely a matter of refining this gesture detection, such an endeavor is a great undertaking outside the scope of this project. That considered, however, the delay in some commands (namely backwards()) is minimal and does not affect the range of motion to a large extent.

Project Goals

The goals of our project, as we set them, were all ultimately accomplished.

First, our system processes the Kinect stream of data into a usable form by filtering out not only visual noise but also by flagging repeated streams of identical commands. Without these algorithms, the internal hardware of the robotics platform would quickly succumb to fatigue and render the parts unusable.

Secondly, we were successful in programming the microcontroller to fully control the robotics platform based on user input. By reverse-engineering several test programs we created our own basic set of methods from which the full range of motion is directly addressable.

Lastly and most importantly, we were able to integrate all of our software and hardware set into a fully-functioning system with relatively minor debugging, given the size and complexity of the system.

Project Timeline

The first few weeks were spent in research and test programs run on a separate Arduino. Initially, we purchased an individual Arduino microcontroller to allow theoretical testing of a robotics platform (that had yet to be chosen). From these sample runs we learned the capabilities and limitations of the board, which we used in researching which parts to gather to build a robot that would communicate properly with the Arduino board.

In our research we considered several possibilities before electing to purchase the Robot Bits shield. We chose the Robot Bits shield for multiple reasons, of which cost and simplicity of function were prominent factors. As our goal was to construct a prototype and funding was limited to our own means, it was essential that we did not have an overcomplicated system, even at the cost of some functionality. We purchased the several parts required to build the shield from the RobotBits website.

The delivery of the parts took much longer than we expected (over 3 weeks). We had to adjust our schedule and work flow to fit with this delay, being at a point unable to proceed without the robot built and working. However, utilizing the test programs available on the RobotBits website, we were able to construct our test programs based on the anticipated command structure of the robot. This proved to be a good decision, as our programs required minor tweaking upon delivery of the platform. Once we received the parts required, we assembled and built the robot with the Arduino Uno board. For this some electrical engineering was required, somewhat different from the usual programming aspect the projects take on. Luckily, Jeff has had some experience and background from working with robotic control boards, and this background helped him tremendously in his construction of the robot.

With the robot finally assembled, we were able to test our programs against the actual platform, instead of outputting success/failure codes to the first board's LED array. We struggled with the first test runs, as the robot seemed to have properly loaded the programs and was running them but it was not responding as expected. The LED lights were responding, however, and a small whine could be heard coming from the robot, despite the lack of movement from the motors. At this point, we were concerned that we had not connected the shield and motors correctly to the board.

After a reassessment, it was determined that it was not a connectivity issue, but rather a lack of power that was preventing our motors from operating properly. We had been running the platform solely on the power gained from the connected laptop. This was not enough voltage to sufficiently power both the board and the motor shield. The robot had an internal battery supply, but the connection to the board would have required proper soldering into a connection adapter on the board. Fear of short-circuiting the board resulted in the use of a wall adapter. While battery power is entirely feasible, our purposes require the robot to be wired to the computer while running and so we decided that a wall adapter was sufficient until the project was completed.

With the robot's new found power, our test programs worked very well on their first run, considering we had not had a chance to debug the code with active motors. We programmed several test functions to control the robot's basic movements, such as forward, reverse, and ninety-degree turning motions. The robot responded almost as expected, and only after a few minor tweaks, our basic control programs were working to perfection.

Since the Kinect side of the project was being handled by a separate team, we elected to delay the implementation of communication between the Kinect and robot until we had perfected every other aspect of the project's feature set.

From here we decided to implement a program that would take inputs from the computer in place of the Kinect to control the movements of the robot. The Arduino board normally runs its programs separately from the computer. In other words, we would write a program in the Arduino C++ based language, load that program onto the board by connecting to the computer, and then the Arduino board would run the program independently of the computer's connection. We needed a program that ran on the Arduino, but continued communicating with the computer to receive input. We developed a program that could communicate with the Arduino and could turn off each individual pin of the Arduino via input from the computer. We integrated this program into our basic motor control program and mapped the keys to the separate functions. Thus, we were then able to control the robot while it was running by pressing keys on our laptop.

All that was left in terms of full hardware implementation was exchanging our keyboard-controlled setup with the actual Kinect software. Our programs were vastly different from one another, and there was a multitude of drivers and installations each team had to install for their computer to communicate with each of our hardware devices. We determined early when discussing with the Kinect software team that it would be tremendously difficult for us to transfer our programs over onto the other's computers to combine the two systems. We decided on using network communications to send messages from the computer connected to the Kinect to the computer connected to the Arduino. Using socket connections, we were able to communicate between the two computers. The Kinect team was generous enough to simplify the raw data streaming from the Kinect to simple character messages when a command gesture was recognized. Our program received these simple character messages and sent the appropriate command to the robot. The robot then responded to the gestures registered.

Project Accomplishments

Specifically, our accomplishments in terms of software were individually very small, but the sum total of their interaction allows this system to be possible.

The Arduino program in combination with the motor shield demonstrates how important it is the abstract the more granular and irrelevant details of the hardware to the user. Not unlike how Java approaches memory management, the Arduino program defines all variables and functions relevant to the robot's range of motion and nothing else. By doing so, any novice may use our functions to develop any program on the platform with minimal effort.

The Client program taught us a great deal about the more niche software packages available in the Java framework, and heavily expanded our understanding of what is possible through a little research and configuration. Furthermore, the communication between two programs in different languages via the use of the Sockets structure was previously not considered a viable method of inter-application communication.

Lastly, the construction of the robotics platform, while not a focus of the overall project, was very helpful in the understanding of the importance of electrical engineering to the ultimate programming of the platform. Without researching the mapping of pins on the Arduino and the attached motor shield, we would have been unable to properly address the movement capabilities of the platform.

Project Results

At the very end of the project, we were able to establish an approximation of the work required from start to completion.

The workload was split very evenly throughout the duration of the project. Despite some scheduling conflicts, we found it very easy to separate our tasks to make all of our established milestones. The two programs and initial Arduino debugging and research was accomplished by Matt Parmelee, while the construction of the robotics platform and maintenance of contact between Kinect and Arduino teams was performed by Jeff Hammel. Between these duties, the total hours spent on testing, debugging and development was 40 hours, which was reduced significantly as a result of the Kinect team working on their development simultaneously.

The lengths of the programs were minimal, as the Arduino board has memory constraints to which the loaded program must conform as well as the abstracted serial interface class in Java. Excluding the serial packages, the sum total of code between our two programs and test programs is approximately 600 lines.

Going into the project our backgrounds in this technology were nearly ideal. Matt Parmelee had a working interest in the Arduino platform and Jeff Hammel already possessed a degree of robotics experience. Experience that would have been of assistance, however, would consist of an understanding of Java packages and the abstracted inner-workings of Java Sockets. Additionally, experience in electrical engineering would have been ideal.

Our interactions with our sponsor, Dr. Chang, were largely positive. Despite some miscommunication at the start with regards to the goals and end product, we found that his direction proved to be very helpful in setting realistic goals and milestones. We feel that our project perfectly conforms to his expectations.

In conclusion, the development process for this project over the past several months has not only been very educational and reinforcing of skills gained as undergraduates in the Department of Computer Science, but ultimately the project was simply enjoyable to complete. We feel we have come out of the experience with a new found interest in robotics and microcontrollers as well as the drive to take on projects of our own design. For this, we would like to thank Dr. Mosse, Dr. Ramirez, Dr. Chang, and the members of the Kinect development team for giving us an enriching experience and a real development opportunity.