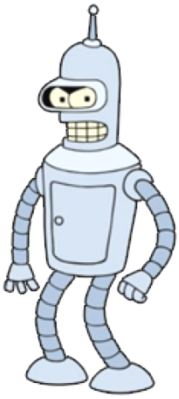
TeleRoboMimicry

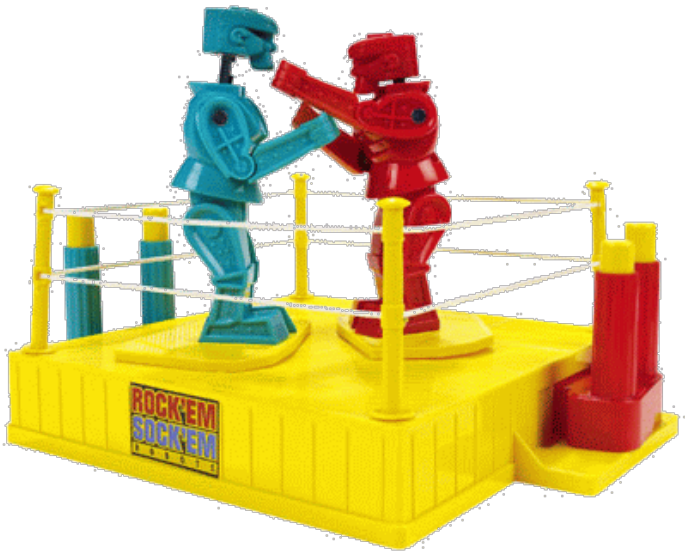
University of Illinois Urbana-Champaign
ECE 395
Spring 2013



Matthew Johnson Eric Badger

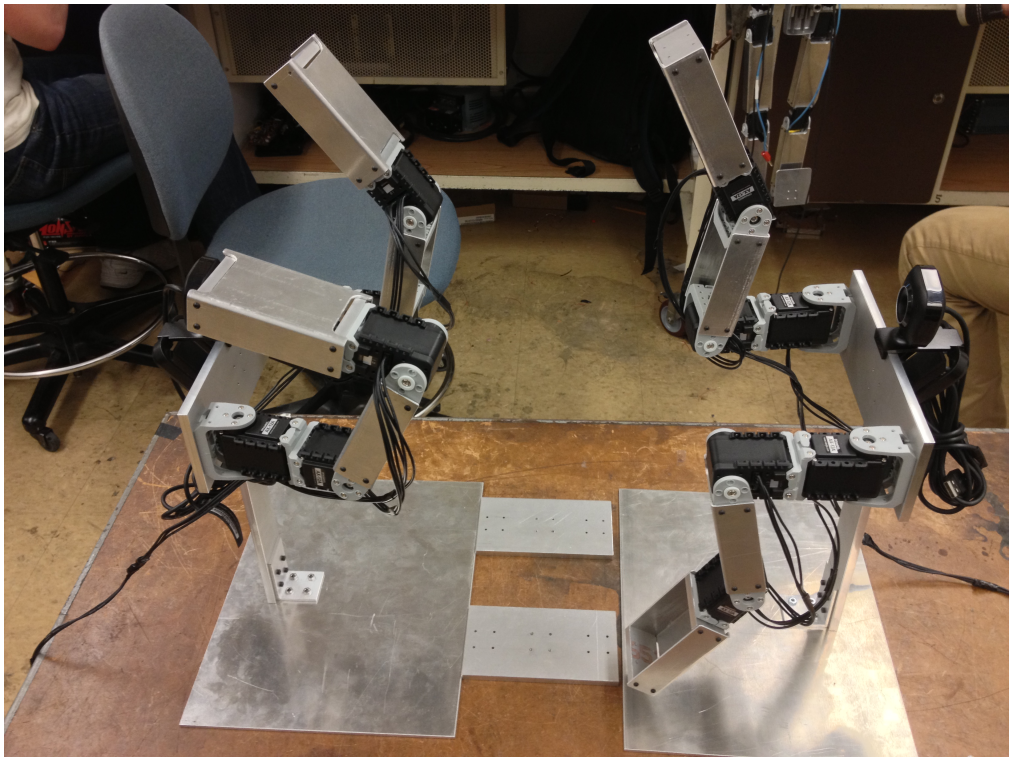
INTRODUCTION

When we first signed up for this class, both of us had quite a few ideas for our project. As we began to narrow down the list one stood out among the rest. This idea was to create a "real" version of Rock 'Em Sock 'Em Robots, a toy that has been around since 1964. The original game (pictured below) featured two robots that punched each other when a player pushed a button



[Rock 'Em Sock 'Em Robots]

Our idea took this a step further. We wanted to design robots that moved and fought based on the arm movement of the player. So, when a player punched the air in real life, the robot he controlled would do the same.



[Our Rock ‘Em Sock ‘Em robots]

During the development of the project, we came to understand that our system had much vaster implications than just a toy. We were essentially designing a robot system that could imitate human movement based on wireless sensors. We began to adapt our design into a more general system, such that it might be used in a number of ways besides fighting robots. To come up with a name for our project, we created a word that evokes the aspects of wireless communication, mechanical robotics, and the emulation of user action. That word is TeleRoboMimcry.

PARTS

MPU6050 6-Axis Accelerometer/Gyroscope

This device is manufactured by InvenSense. It has a self-contained 3-axis accelerometer and 3-axis gyroscope. This is similar to the type of device that is used in smartphones to detect the angle at

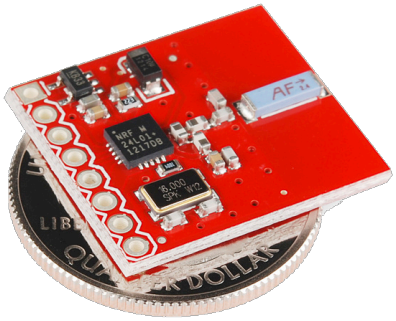
which it is being held in order to rotate the screen. In our case, we use the MPU6050 to determine the angles of the human's upper and lower arm segments relative to gravity. When powered on and set up, the MPU6050 reads values from the accelerometer and gyroscope and stores them in a number of hardware registers. These registers can be accessed via an I2C (Inter-IC) serial protocol. The I2C rate used for this device is I2C fast mode, which can communicate at 100Kbit/s. The chip also has an onboard Digital Motion Processor (DMP), which can be configured to calculate yaw/pitch/roll angles, quaternions, and more from the accelerometer/gyro data. Due to the details as to the configuration of the DMP not being freely available (InvenSense requires an NDA), we did not use the DMP, and did the necessary mathematical calculations on the microcontroller ourselves. For further documentation about MPU6050 usage, see http://www.invensense.com/mems/gyro/mpu6050.html.

[MPU 6050]

NRF24L01+ Transceiver

This device is manufactured by Nordic Semiconductors. It is a 2.4Ghz radio frequency transmitter/receiver pair. We use them to communicate between the boards worn by the user (referred to as the “Satellite” board) and the board used to control the robot servos (referred to as the “Based” board). This chip has a plethora of features and modes, all of which can be configured over a SPI (Serial Peripheral Interface) protocol. Features include multiple pipelines, a packet addressing system, variable operation frequency, cyclic redundancy check (CRC), and more. The two modes of interest to

us were the normal RX and TX modes. When the device is put into RX mode, it will receive data on the frequency channel for which it is configured. It automatically matches the address of incoming packets to its own configured address and discards packets that don't match or fail the CRC (Cyclic Redundancy Check). Packets that are successfully received are stored in an on-chip FIFO buffer that can be configured to send an interrupt to the microcontroller upon receiving new data, which can then be read over SPI. In TX mode, a transmit FIFO buffer can be loaded with a packet over SPI, and then transmission can be triggered with a Chip Enable line. The transmission data rate over RF between transceivers is 1 Mbit/s. Documentation for this transceiver is at http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01.



[nRF24L01+]

PowerCell

This is a small board manufactured by Sparkfun Electronics that ontains a Lithium Polymer (LiPo) cell charging circuit, a 3.3V to 5V boost converter circuit, as well as a switching regulator. We use these boards, along with single cell 1000mAh LiPo batteries, to provide power to our Satellite boards. Sparkfun has a PowerCell tutorial here: https://www.sparkfun.com/tutorials/379.

[PowerCell + LiPo Battery]

Dynamixel AX-12a Servos

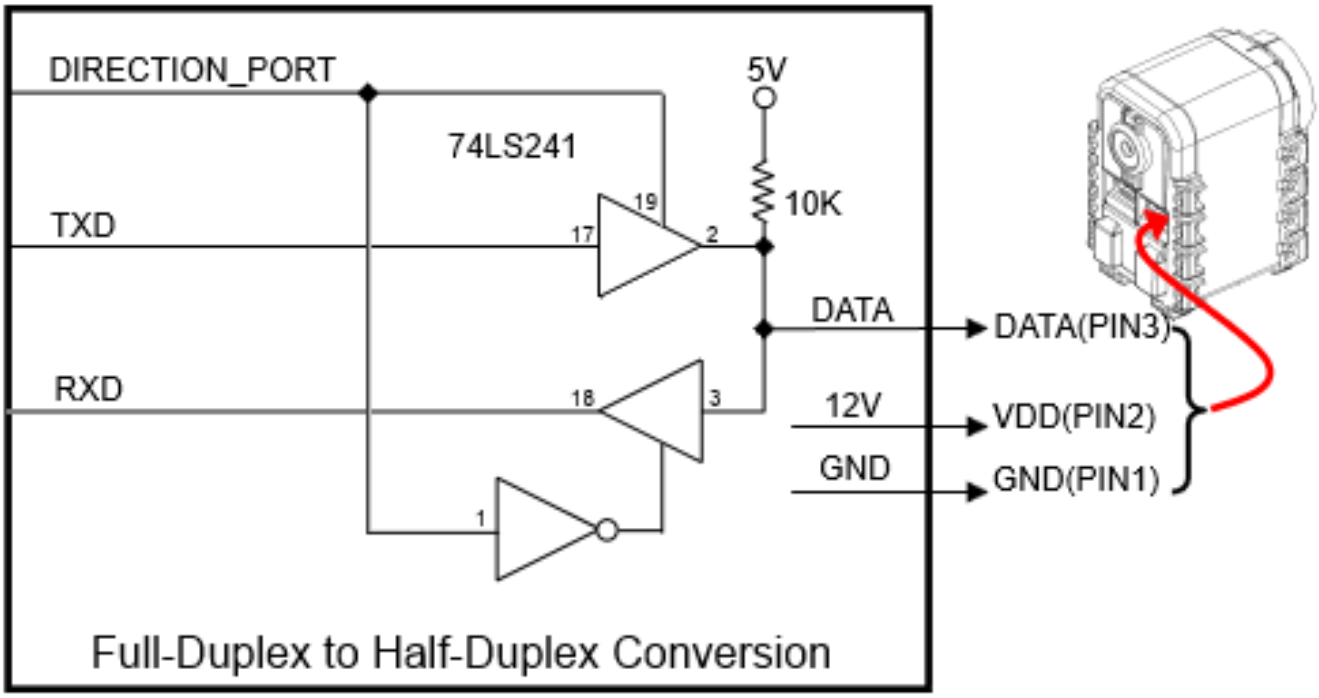
The servos we used for the robots are made by ROBOTIS. These servos are a bit different from the ordinary kind because they have a tiny built-in microcontroller. So, instead of driving them with pulse width modulation (PWM), commands are issued to the servos over a serial protocol and among other things, the onboard microcontroller does the PWM itself. Because this is a smarter-than-average servo, the AX-12a can do things like monitoring its own torque and temperature, and it can shutdown to prevent damage if these values get too high. The servos each have a set of registers for reading and writing the values mentioned, along with much more settings and data. The AX-12a runs on 9-12V, and can provide torque up to 15.3 kg-cm (1.5 N-m).

Communication with these registers involves an asynchronous serial protocol, with a baud rate of 1Mbit/s. The servos are all connected in parallel with a half-duplex setup, meaning that there is 1 wire for power (12V), 1 wire for ground, and 1 wire for serial data. In a half-duplex setup, transmission and receiving of serial data occurs on the same line, as opposed to the more common full-

duplex setup, which has a separate line for transmit and receive. Since, all of the servos are connected in parallel, an addressing system is used to determine which servo should respond to a command. All servos have a configurable address register, and only respond to serial commands that are prefaced with an address corresponding to their own. There is also a reserved address (254) for which all servos will respond.

The LPC microcontroller on our Based board used to communicate with these servos has a full-duplex UART, so we needed some additional logic to convert the full-duplex serial to half-duplex. This can be done with a couple tri-state buffers, controlled by a direction pin driven by the microcontroller on the Based board. The direction pin toggles to indicate whether the Based board is transmitting or receiving. Documentation for the AX-12 series can be found at http://www.crustcrawler.com/products/bioloid/docs/AX-12.pdf.

[AX-12a Front and Back]

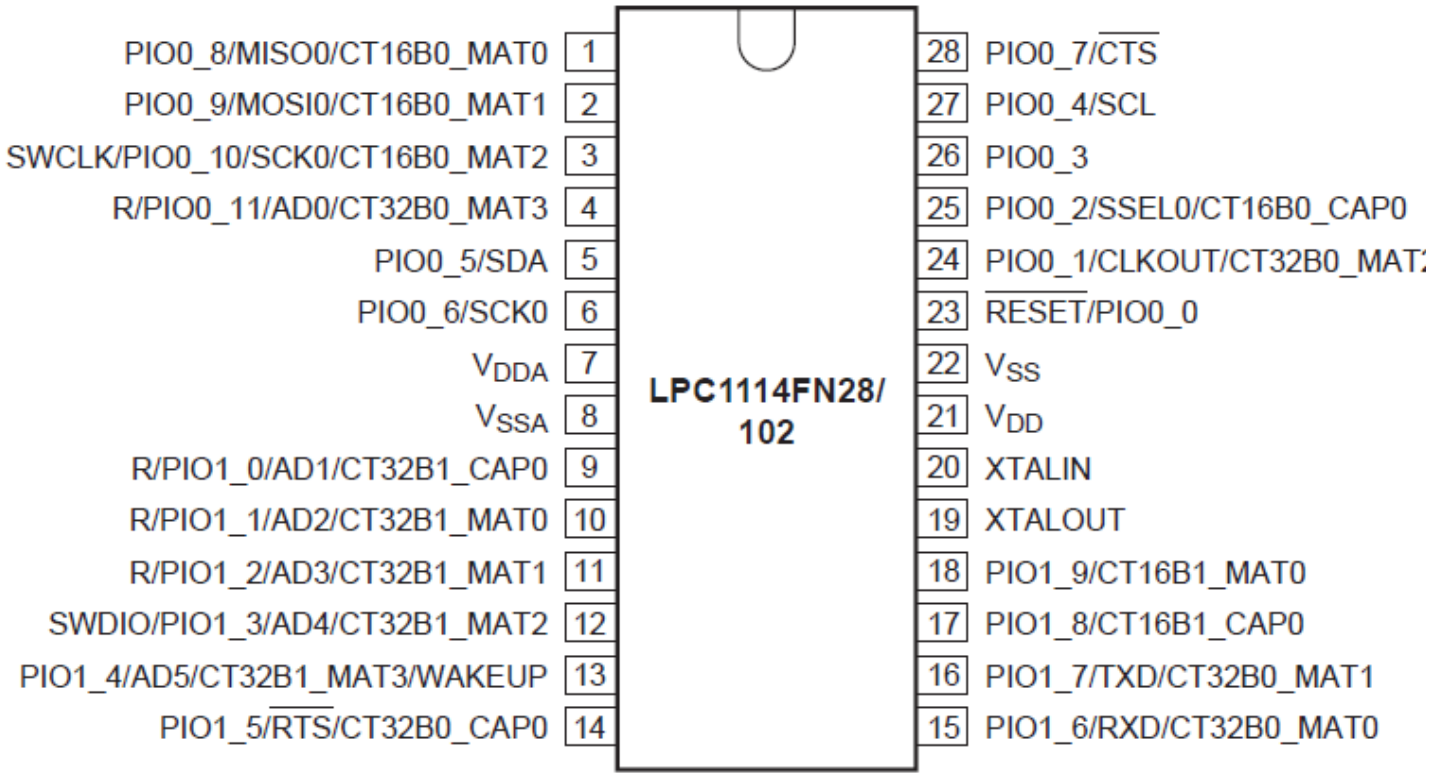


[Duplex logic]

LPC1114/102 Microcontroller

This is the microcontroller we used for both the Based and Satellite boards. It is manufactured by NXP (formerly Philips Semiconductors). The LPC1114/102 (simply "LPC" for future references) is a low cost 32-bit microcontroller based around an ARM Cortex-M0 processor, operating at 48MHz. Peripherals on the LPC include a UART, I2C (Fast Mode), 2 SPI (SSP) Interfaces, 4 Timers, and up to 42 GPIO pins. We decided to use this chip because of its variety of peripheral interfaces, its very low power design, and its cost (< $2.50 a chip). We ended up going with the 28 pin DIP package (pictured

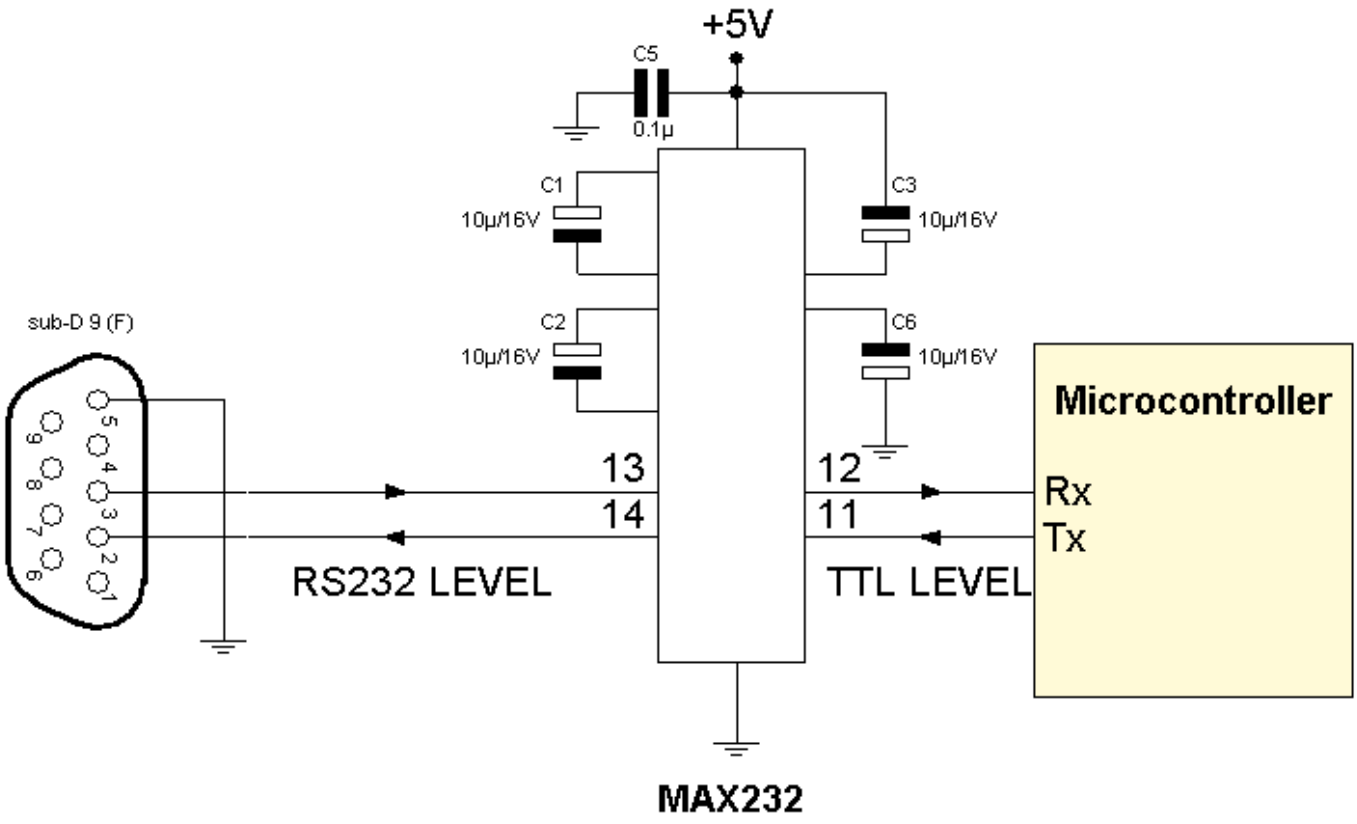
below). For documentation, see http://www.keil.com/dd/chip/6526.htm.



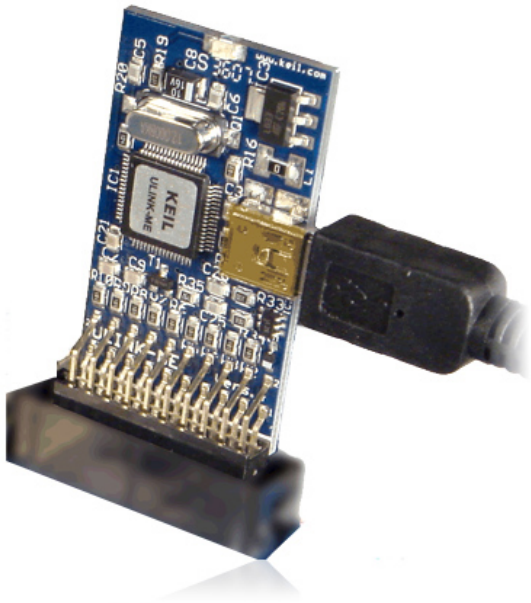
[LPC1114 DIP28 pin diagram]

Miscellaneous Parts

Our boards contained a variety of other miscellaneous parts. These included components such as resistors, LEDs, and pin headers. The aluminum parts of the robot (the arm segments and the base/stand) were designed by us and machined in the ECE Machine Shop in Everitt Lab at the University of Illinois Urbana-Champaign. We also built a USB serial monitor (using a USB-serial cable and a MAX232 chip) so that we could debug over serial using port monitoring software. One other thing that we used was the Keil ULINK-ME USB-JTAG Interface Adapter. This device works with the uVision IDE to both flash and debug programs on the LPC. Documentaion for the ULINK-ME is here: http://www.keil.com/support/man/docs/ulinkme/default.htm.



[Serial logic level conversion circuit]

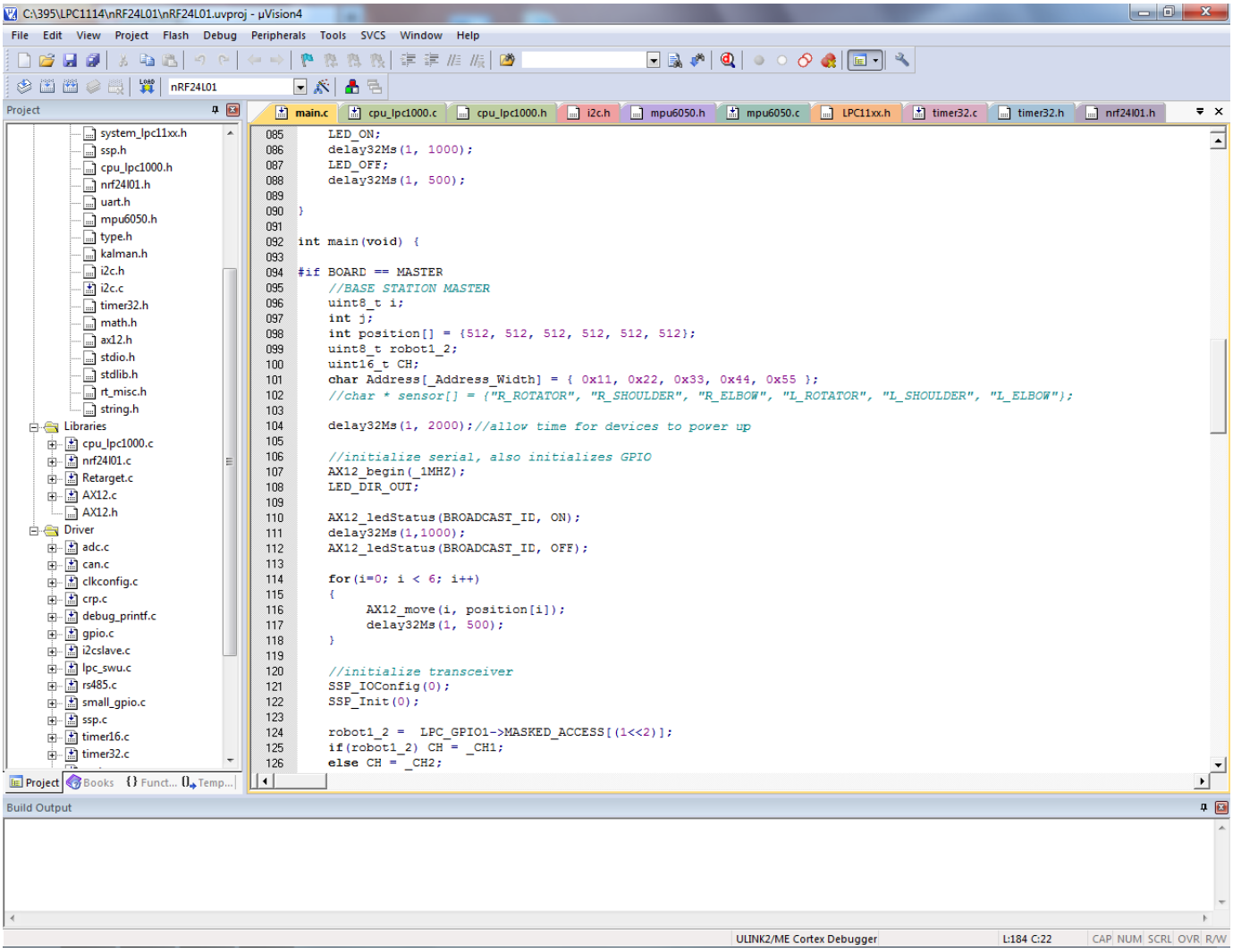


[ULINK-ME programmer/debugger]

SOFTWARE

uVision

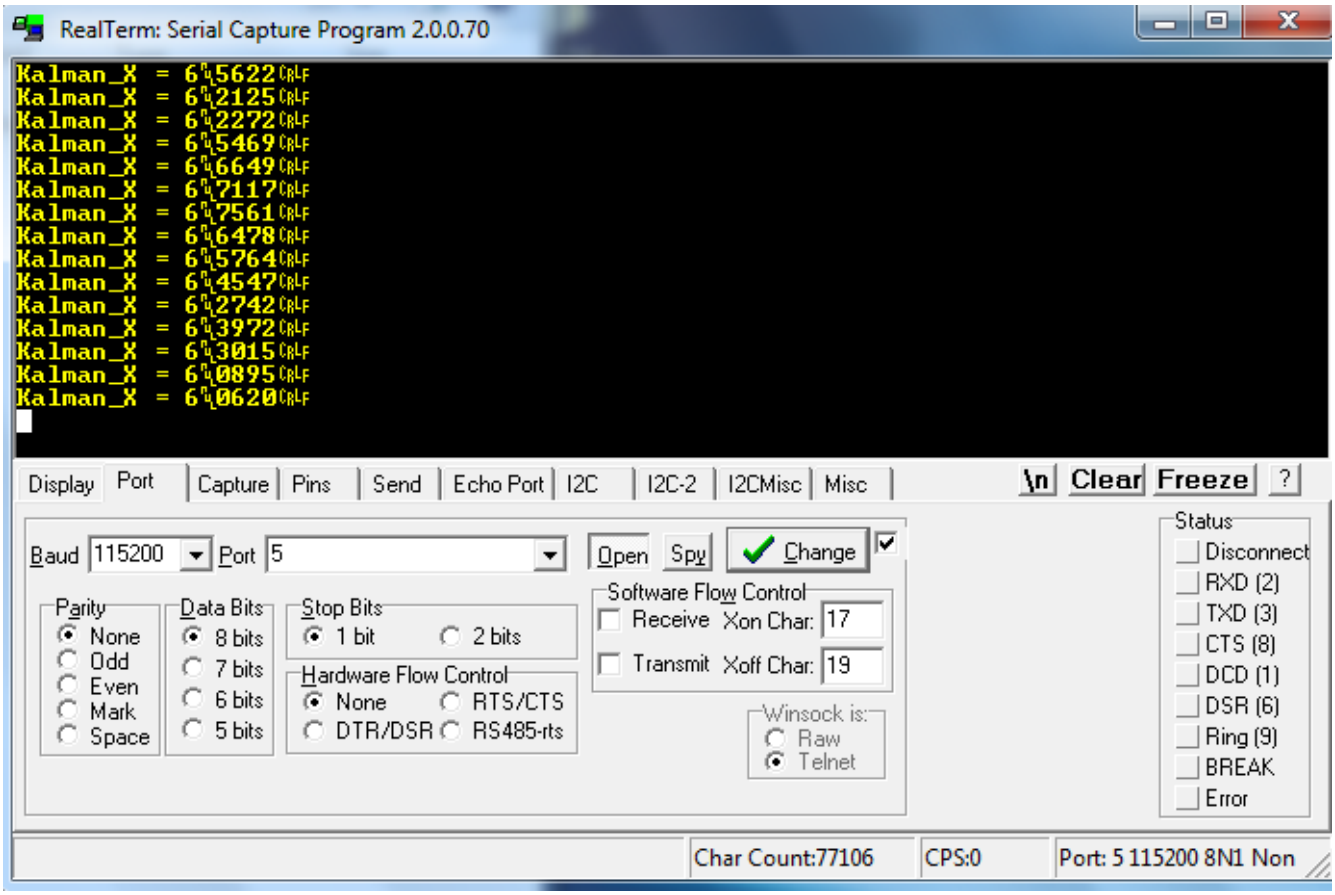
This software by Keil is an Integrated Development Environment (IDE) that has powerful tools for embedded programming. It has compilers/linkers/assemblers for a variety of chips (including the LPC1114/102). It has a program flasher and debugger that work with the ULINK-ME. And it also has tools to analyze clocking, power, and nearly all peripheral interfaces including GPIO. Keil's documentation for uVision is at: http://www.keil.com/uvision.



[uVision IDE screenshot]

RealTerm

This software is used to analyze data streams. We used it in conjunction with our USB serial monitor to analyze serial debug output from our LPC chips. The USB shows up as a COM port, and analyzing the data stream is as simple as selecting the appropriate baud rate, port number, and data framing/flow settings. We set up our debugging serial line on the LPC UART to be 115200 baud, 8 data bits, 1 stop bit, no parity, and no hardware flow control. Download and documentation for RealTerm can be found here: http://realterm.sourceforge.net/index.html

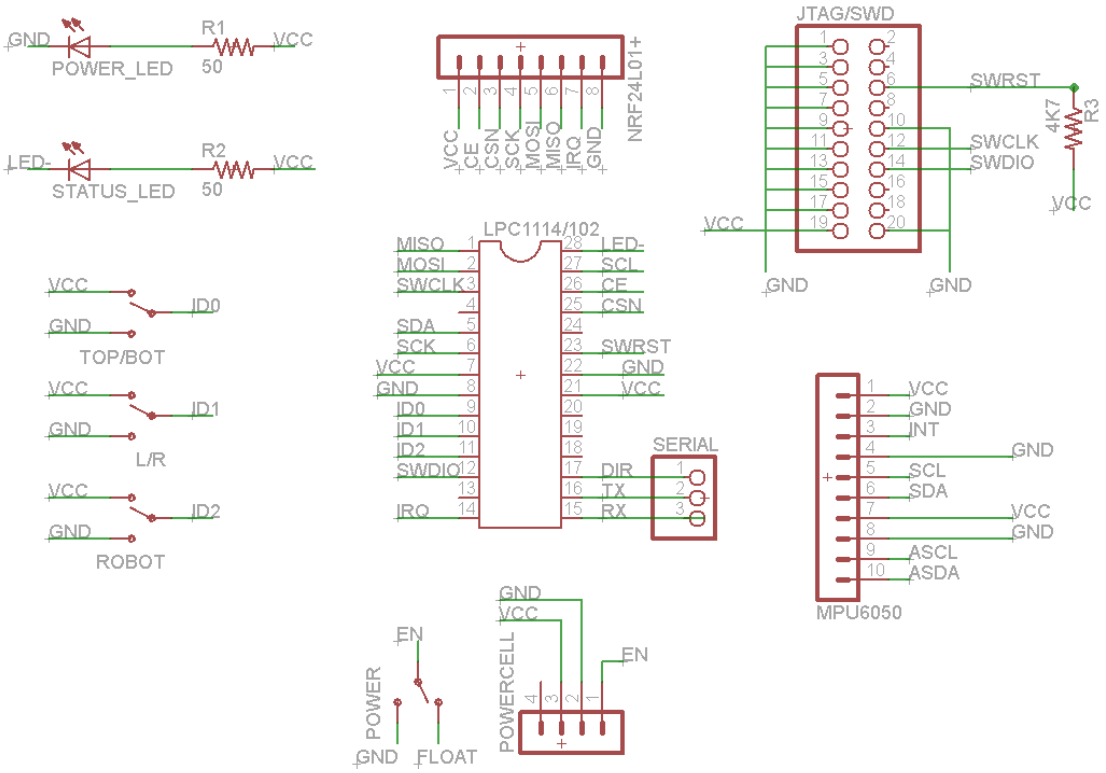


[RealTerm screenshot with our configuration]

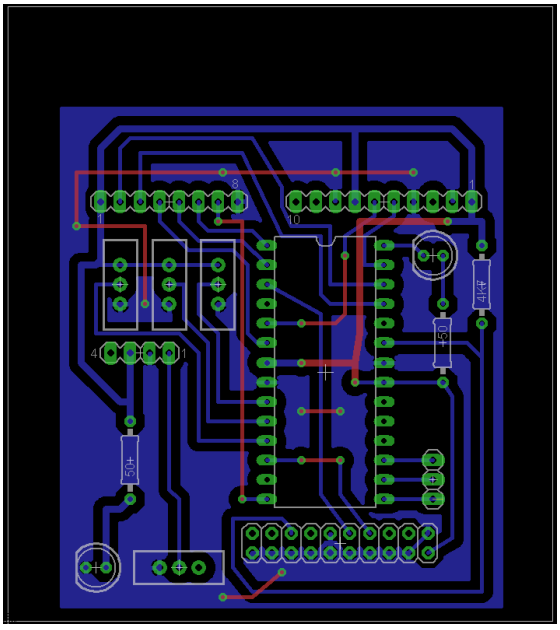
Eagle

This software is made by CadSoft. Eagle is a printed circuit board (PCB) design suite. Users can import libraries of circuit components from manufacturers, design their own parts, create a circuit

schematics, lay out components on a board, route wiring between components, generate files for automated PCB manufacturing systems, and more. Some very good tutorials for Eagle can be found at http://www.jeremyblum.com/category/eagle-tutorials.



[Eagle screenshot of our PCB schematic]



[Eagle screenshot of our PCB]

GitHub

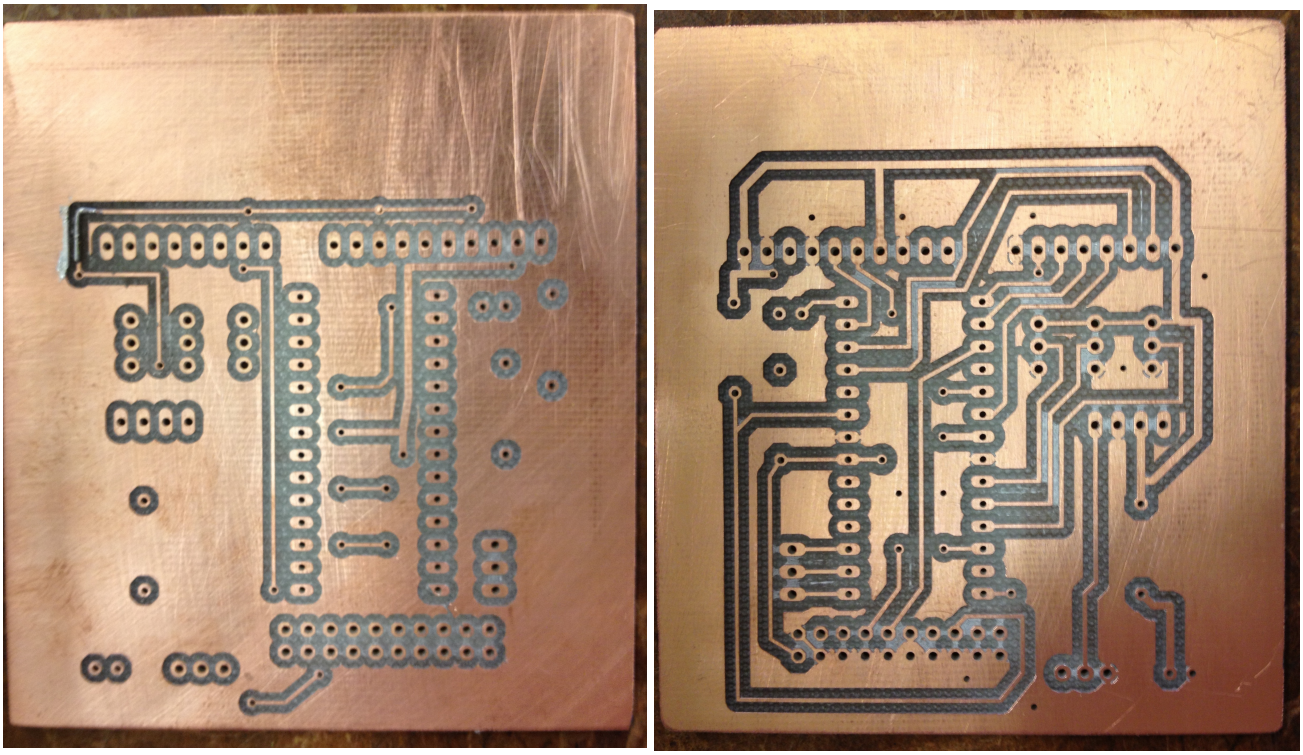
Git is a version control software (VCS) used to maintain code. A VCS allows programmers to keep several versions of their code along the development cycle. For instance, if a programmer changes something and ends up breaking their code, a VCS can allow them to revert back to the last time the code worked. A VCS is also helpful when multiple programmers are working on the same project, as it is able to merge changes made by different users. So Git was a very useful tool for us, especially since we had multiple files for each device and driver with which we were working. GitHub is an extension of Git that allows users have online space for stored and presenting their code. GitHub is designed for collaboration as well, with user profiles, built-in issue tracking, branch visualization, code forking, pull requests, wikis for documentation, and more. A detailed online book about Git is here: http://git-scm.com/book. An interactive tutorial can be found here: http://try.github.io/. The Git repository for *our* code will be hosted here: https://github.com/Mrjohns42/telerobomimicry.

BOARDS

PCB Manufacturing

As mentioned above, we designed our PCB in Eagle after first prototyping our circuit on a breadboard. This was the first PCB that either of us had ever designed, but Eagle is very powerful and pretty easy to learn. We designed our PCB to be multifunctional. Depending on which parts are soldered and what software is flashed on the LPC, the PCB can function as either a Based or Satellite board. Laying down wire traces is typically the hardest part, because wires on the same layer cannot cross if they are not meant to be connected. We ended up creating a 2 layer PCB with only minor difficulty. After that, we generated Gerber files from the board layout. Gerber files are standardized descriptions of all the layers of the PCB, and are read by PCB manufacturing systems to actually make

the PCB. There are a variety of methods to manufacture PCBs, but our PCBs (10 of them) were milled on an automated milling machine in the ECE Service Shop in Everitt Lab at the University of Illinois Urbana-Champaign by Mark Smart. All of the parts were thru-hole parts, which allowed us to solder the boards ourselves pretty quickly. The Satellite boards were mounted onto elastic arm bands using some hot glue. Depicted below is the milled PCB.

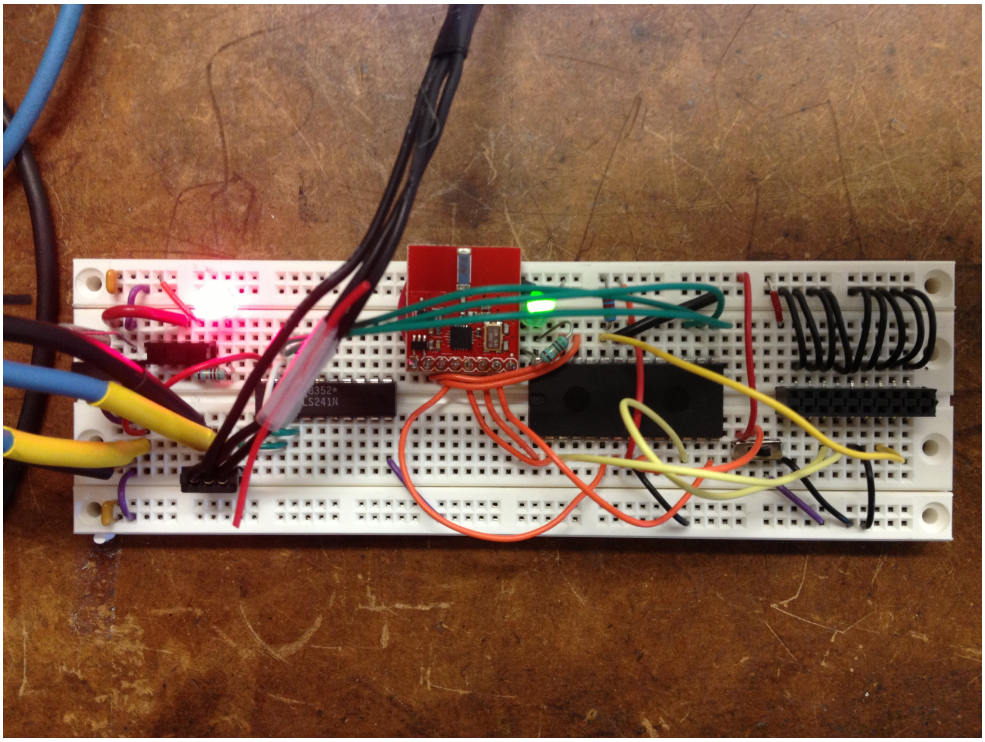


[Milled PCB front and back]

Based Board

Each robot has its own Based board that sits at the base of it. The Based board has an LPC, an NRF Transceiver, a PowerCell with LiPo battery, header for the ULINK-ME, switches for robot selection, and header for serial servo control output. The role of the Based board is to receive position data from the Satellite boards and use that data to control the robot's servos. Since the PCB we designed didn't include the logic for full to half duplex conversion, we ended up using a breadboarded version for our demonstration. The breadboard has all the circuitry of the PCB, as well as the full to

half duplex conversion logic. Instead of running on a LiPo battery, our breadboard took 12V and 5V from a desktop power supply. The 12V is used to directly power the servos. The 5V is used to power the full to half duplex logic, and a voltage regulator was used to step the 5V down to 3.3V to power the microcontroller circuit.

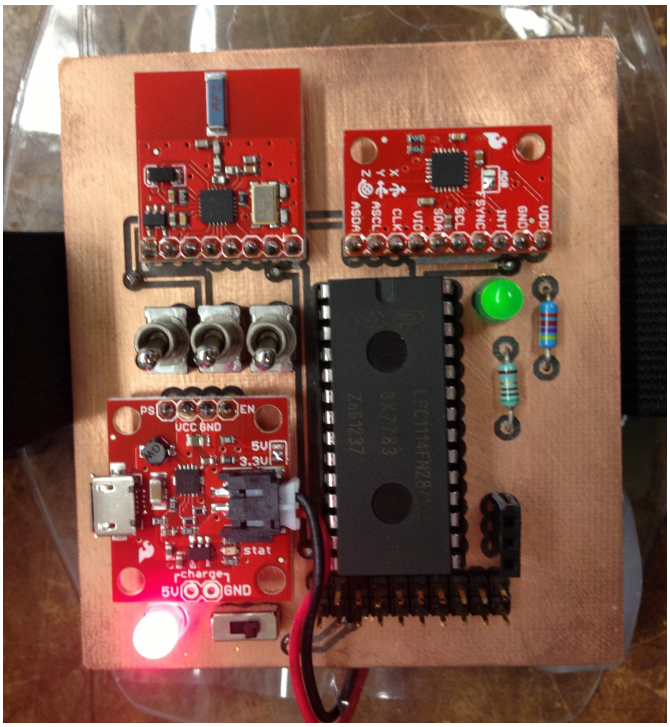


[Breadboarded Based board]

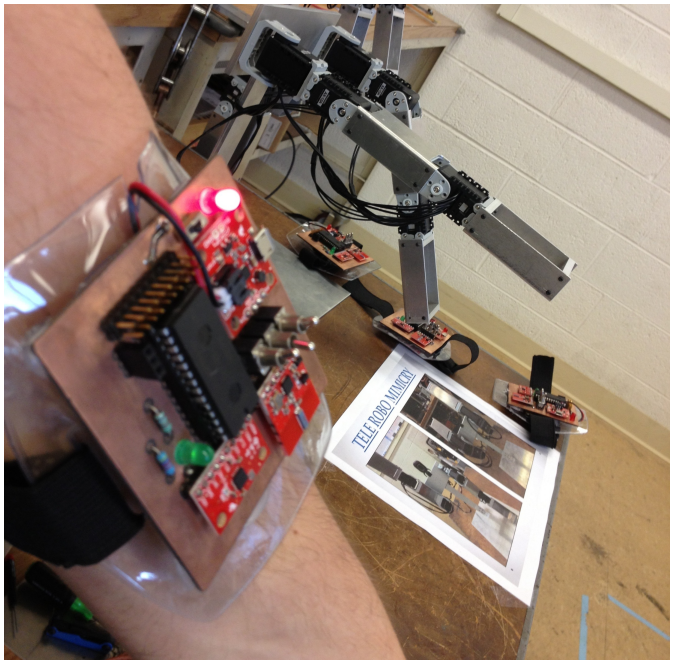
Satellite Board

Each robot was associated with 4 Satellite boards, one for the bicep and forearm of each arm. The Satellite boards each have an LPC, an NRF Transceiver, an MPU Accelerometer/Gyroscope, a PowerCell with LiPo battery, header for the ULINK-ME, switches for robot and arm segment selection, and header for serial debugging output. The role of the Satellite board is to sample the MPU repeatedly, applying Kalman filtering to the values. When the Based board sends a request to a Satellite, it triggers an interrupt in the LPC, which responds by transmitting its current calculated value back to the Based board. To make the board wearable, we found some armbands online that contain a

plastic sleeve, typically meant for holding ID cards. We hot glued our Satellite boards to the top of the plastic sleeve, and stored the LiPo battery inside the sleeve.



[Satellite board head-on]



[Satellite board mounted on armband]

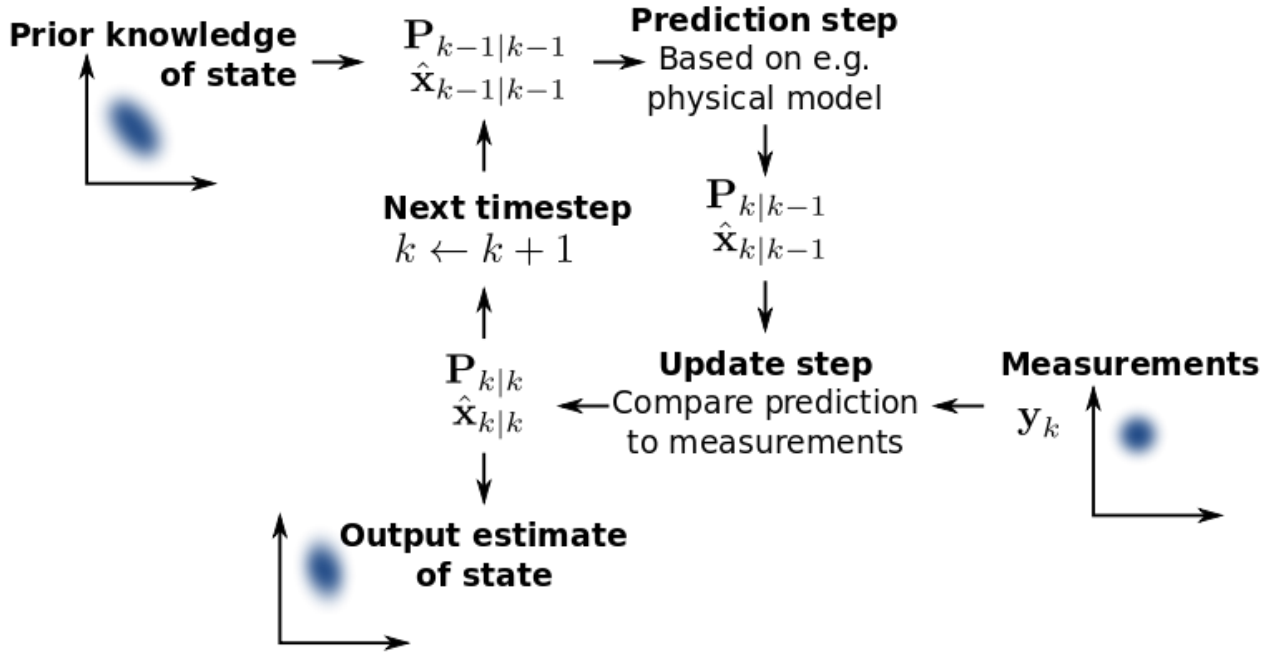
CONCLUSION

Operation Description

The user wears 4 Satellite boards, one for each of the left and right forearms and biceps. When the Satellite board is turned on, the LPC first initializes all the devices on the board. It then enters a loop that communicates with the MPU-6050. Using an I2C protocol at 100Kbit/s, the LPC reads from the MPU registers that contain the current accelerometer and gyroscope values. We make use of the acceleration (via the accelerometer) about the MPU's x axis and stabilize the value using the angular velocity (via the gyroscope) about the x and z axis. A timer is also used to track elapsed time between measurements. As another part of the code's loop, we then Kalman filter the stabilized values. A Kalman filter is an algorithm used to reduce noise in measurements. The algorithm works by using known laws of the system's dynamics (in our case, the laws of rotational kinematics) along with previous measurements to estimate the next measurement, and using the difference between the estimated and actual next measurement as feedback to better hone the filter. It does this by weighing estimates with a smaller variance from the actual measurement more heavily in future estimates. A good intro to Kalman filtering can be found here: http://bilgin.esme.org/BitsBytes/KalmanFilterforDummies.aspx.

Kalman filtering greatly reduces the impact of noise and drift in the values for arm position measurements. The LPC's sample and filter cycle repeats until the NRF Transceiver on the Satellite

receives a request packet, triggering an interrupt on the LPC.



[Diagram from Wikipedia depicting Kalman filtering]

Once the Based board is powered on, it initializes the devices on the board as well as the servos. The Based board then starts communication by cycling through the addresses of the Satellite boards one at a time and sending requests to each of them asking for arm position data. The request begins by switching the NRF Transceiver on the Based board into TX mode. Over SPI, with a clock speed of 24 MHz, the LPC loads the TX FIFO with the request packet, and toggles the Chip Enable line to initialize transmission (occurring at 1 Mbit/s). It then switches the NRF Transceiver back into RX mode to await a response.

The request transmitted from the Based board arrives at the NRF Transmitter on the Satellite board, which triggers an interrupt on the Satellite board LPC to send back a response. Inside this interrupt routine, the Satellite LPC (once again using SPI with a 24MHz clock) reads and verifies the request from the Based board. It then switches the Satellite NRF Transceiver into TX Mode, loads the TX FIFO with the current arm position angle, and toggles Chip Enable to send the data back to the

Based board (at 1 Mbit/s). The Satellite switches its NRF Transceiver back to RX mode, and goes back to Kalman filtering MPU data.

Then, the Based board receives the response from the Satellite board, it reads the transmitted arm position data from the RX FIFO on the Based board's NRF Transceiver. It uses this data to update the position of the servo that corresponds to the Satellite from which the data originated. Communication between the Based board LPC and the servos occurs at 1 Mbit/s. The servo then sends back a response packet, which may include error messages if something went wrong. After this, the Based board continues cycling through the rest of the Satellite addresses repeatedly.

The overall effect is that the robot mimics the exact arm position of the user in real time. When the user moves their arms—be it to punch, wave, or dance—the robot does the same. Telerobomimicry has a vast variety of applications.

Here is a link to a live demonstration of our system:

http://youtu.be/hO_E-P2LmRk

Speedbumps

There were a few noteworthy problems that we encountered while bringing our system to life. These "speedbumps" can be categorized into three separate issues: start-up timing, synchronization, and missing parts.

Initially, communication between the MPU device and the LPC chip would only work in debug mode with the ULINK-ME debugger. In regular execution, the MPU just didn't respond to any I2C packets. This was a very strange issue, because debug mode runs the same code as in regular execution. We examined everything from register initialization to the debugger's influence on clock speed, but couldn't find the issue for several days. After a bit of trial and error, we were able to deduce that our MPU (along with all our other peripherals) require a certain amount of set-up time after being powering up before it can respond to I2C commands. During regular execution, the board is reset by toggling power, but in debug mode, the board is reset by the debugger. Because of this, the

initialization functions that began immediately after power-up were failing. This left the chips in an unresponsive state. The issue was finally resolved by adding a simple delay between the time LPC starts up and the time it calls the initialization functions.

Another issue was the synchronization between the Based board and the Satellite board. The NRF transceivers can be set to be in either TX or RX mode. Initially, all boards are initialized to RX mode. When a board needs to transmit, it switches to TX mode, loads the data, triggers the transmission, and then switches back to RX. However, both RX and TX require some set-up time upon entering the mode before it can execute. Forgetting to account for this set-up time, our Satellite board would often respond to requests and transmit data back to the Based board, before the Based board had completely transitioned into RX mode, causing the transmitted packet to be lost. This resulted in the Based board being stuck waiting for data that had already been sent. Meanwhile, the Satellite board gets stuck waiting for an ACK on its transmission from the Based board. This leaves the system in a deadlock with both boards waiting for each other forever. Once we understood the problem with the deadlock situation, it was easy to fix, but it was difficult to diagnose why we were deadlocking. To fix the problem, we made two changes. First, we added in a small delay after a packet is received before it transmits a response, giving the other board time to switch modes. The other change we made was to add a timeout to the transmission and receiving subroutines, so if a packet is dropped, the system doesn't get stuck in deadlock.

The final major problem that we ran into was more of a physical problem rather than an implementation problem. A few days before the demo, while we soldering all of our boards together, we found out that two of our NRF Transceivers had gone missing, be it accidentally or on purpose. We ordered new parts to be shipped overnight, but the company did not place our order until the following day and the parts did not arrive on time. Because of this, we were unable to build our last two satellite boards and so we only had one robot available during the final demonstration. We received our parts a

day late and we now have both robots working.

Possible Applications

Our system and overall concept of TeleRoboMimicry has many potential applications. There are many situations in which a human is needed to complete a task, but the mechanical robustness of a robot is preferable. These situations are well suited for applications that extend our design to the full human body. Examples of such applications would include activity in quarantined areas, chemical/nuclear waste or fallout, and even space exploration. These areas are dangerous to humans, but could be more easily navigated by a robot under human instruction.

Another example of a potential use is robotically assisted surgery. Some types of ultra-precision surgery can be difficult for humans to perform, but easier for robotic instrumentation. Additionally, there is the added potential of having a doctor perform surgery on someone halfway around the world. Without actually having to be present or travel to perform surgery, medical specialists could treat more patients and treat them more efficiently.

Our system can also be used in defense. Having unmanned droids/drones that can be controlled by actual humans would give you the intuition and decision making of an actual trained human paired with the durability and expendability of a robot. This technology could revolutionize the way wars are fought and reduce the amount of casualties on either side of the battle. Imagine a real-life Iron Man.

Additionally, this system could be used as a type of input for a computer or a game console. It could be used in an interactive system where a human controls input with his/her own body, instead of a controller. This would be a similar input to what is collected by the Xbox Kinect, but created with physically worn sensors instead of computer vision. Similarly, this type of input could be useful in motion tracking for movie/game production.

Finally, this technology could be used in some capacity to implement high-tech prostheses. In the prostheses, sensors that detect electrical activity in the brain or muscles could be used to control

replacement robotic limbs. If this could be perfected, it would be just as if they never lost their limbs at all.

Final Remarks

We both really enjoyed this opportunity to take ECE 395 during our last semester as undergrads at UIUC. The kind of experience that one gains in this course is directly applicable to the skills one needs to succeed after college, but it is also the kind of the experience that cannot be taught in a typical lecture. ECE 395 inspired us to be self-motivated learners, to think critically, to solve problems pragmatically, and to be comfortable exploring the unknown.

Special Thanks

Zuofu Cheng - ECE 395 TA

Dr. Lippold Haken - ECE 395 Instructor

Mark Smart - ECE Service Shop

David Switzer - ECE Machine Shop