Portland State University

INTELIGENT ROBOTICS II

ALBERT EISNTEIN

Winter 2011

Abstract

This report presents an overview of the work completed by the Albert Einstein group members in the Winter 2011 Intelligent Robotics II course. The report presents documentation the group has generated and collected so that future groups may benefit from our work. It is also details some of the challenges the group has encountered during the course work.

Group Members and roles

Nguyen, Tin……..Hardware components

Shaat, Gehad……..Development Process Summary

Schaeffer, Ben……..Development Process Summary

Truong, Jessie……..Einstein's animation

Upperman, Nathen……..Hardware components

Contents

Table of Figures

Introduction

The Albert Einstein robot consists of a puppet-like humanoid which is mounted on a motorized base built of wood. There are 13 servos connecting Einstein's wooden body parts which permit electronic control his eyebrows, eyes, both arms and legs. The base is controlled by an NXT connected to three ultrasonic sensors and Hitec motor controllers for the wheels. A camera was added and mounted on top of Einstein's head which was connected to a laptop which operates as the "Robot Brain".

The goal for this project was to one, resurrect a damaged robot, two, create OpenCV-based tracking routines that would control Einstein's motion and behavior, and three make well-designed and documented software components that would formally remain on the RAS repository and be available to future groups.

Tools used

- IDE
    - Visual Studio 2008 (developing C++ application)
    - RobotC (developing NXT application)
    - Appinventor (developing android application)
- Libraries
    - OpenCV 2.2b4766
    - CUDA toolkit
    - NVIDIA Performance Primitives (NPP) library
    - GPU Computing SDK
    - TBB (Threading Building Blocks)
    - .NET C++ Framework
- Software
    - tortoise SVN
    - Google sites
    - Repository
- Hardware
    - NXT
    - Maestro Servo Controller

    - DC motors with Hitechnic Controller
    - Self-built Power Distribution System
    - 16 V battery
- Other
    - Einstein's Knowledge!

Accomplishments

- Bluetooth API and implementation
- Servo controller application (animation)
- Android application (robot remote control)
- Object and human detection application
- Efficient Power system
- Integration of all previously mentioned component into einsteinBrain
- CUDA & TBB base program

Future of the project

- Use neural network to improve Einstein's Artificial intelligence!
- More balanced weight
- Improve power efficiency
- Add wireless camera
- Add more sensors/ stereo camera
- Replace the neck motor with servo (or use a stepper)
- Use of CUDA & TBB to improve speed
- Replace NXT with motor controller
- Build a PID for neck motor
- Build Buck for Power Supply
- Faster the motor for omniwheel implementation(stepped for motor control)
- Counterweight on back of the Einstein (he is front heavy causing the back to flip up every time Einstein stops hard
- Use Einstein’s eye changing characteristics for some logic state

Image Processing

- We did the following processing on images
  - Find colored objects
  - Detect how many beats per minutes for a drumstick
  - Face detection with area calculation

Hardware Components

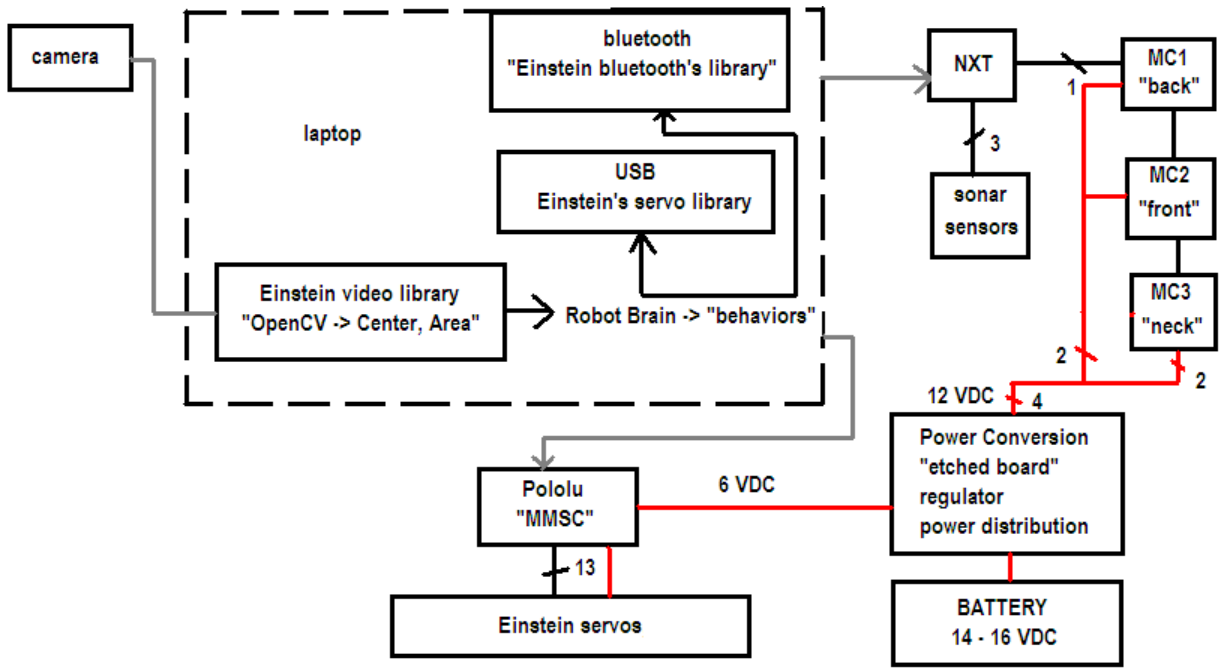


Figure 1: Block Diagram

The Power System

There were key points that must be noted here about the particular power system that was designed for Einstein. Einstein had originally had multiple switches and power supplies, meaning multiple types of batteries and an AC power supply. This type of setup is not easy to follow, and neither practical nor professional, although it was reliable. From an Electrical Engineering perspective we wanted to converge all of it into one battery supply for many reasons:

- Easy to troubleshoot (we do not have to find out which power supply has failed)

- Looks better (there aren't any random batteries hanging everywhere)
- There is only one battery type to charge (multiple types of batteries would require different types of chargers
- High initial cost, but long term benefits (The initial cost was quite expensive as mentioned later, however, the system can be used multiple times and for a longer period of time)

- Easier for the next person to use (do not have to figure out where all the other battery supply systems go, nor have to reroute the power system)

- It is common practice to have a high voltage dc source stepped down to source other systems , such as in a car or in a portable radio, where multiple power supplies would be impractical (learning how to do this now will help me in the future)

One company mentions that, ‘‘…where 12V DC from a car battery must be stepped down to 3V DC, to run a personal CD player; where 5V DC on a personal computer motherboard must be stepped down to 3V, 2V or less for one of the latest CPU chips…,” (Jaycar Electronics 1). As you can see, this process is normal and it happens in practice quite a bit.

The Battery

There are many batteries to choose from and since we chose to build the electrical system from one power source, we wanted to ensure we had high current that could supply all of the motors. Nathen decided to use the Lithium Polymer type of battery even though they were more expensive than a lead acid. He purchased these batteries and the charger at his own expense, so as to use them later in other projects and to preserve the Robotics Fund. The total amount including shipping was about $160.00. The batteries themselves were about $40.00 a piece (2 were used in parallel) and are the same type of batteries used in laptops and other appliances of the like. These are much different then the lead acid batteries, but these are his reasons for this purchase:

- Supplies constant current even when the battery is getting low
- Much lighter than lead acid
- Take up less space
- Higher voltage
- Hold their charge longer

- Offer the most power
- Balanced Charging (each plate is checked for voltage and charged evenly)

Figure 2: Lithium Polymer battery

A few downfalls are that these batteries can be easily overcharged causing damage to the cells, which is why a special charger is needed, and the batteries have a tendency to charge higher than the rated charge. This is actually why we have had so many problems with the power system. Mostly, because the batteries were at such a high voltage that the regulators had a problem dealing with the heat when the batteries are fully charged and due to the high current that the system was pulling. The particular charger Nathen purchased has a few safety features and can charge different types of batteries.

Figure 3: Charger

Power Conversion

The system needs to be converted from the 14 to 16Vdc to two separate power systems: one with 12Vdc to power the 4 motors on the base and the motor on Einstein's neck and his eyes (See Einstein's Eyes), and the other with 6Vdc to power all of the servos. This was primarily accomplished by

using multiple voltage regulators. Two of the three motor controllers on the base each have two 12Vdc motors connected, which pull about 200mA per motor on a medium load. Thus each motor controller has about 400mA load, plus a few milliamps to power the controllers themselves. The last motor controller only has one of these motors connected to it and to power the controller itself.

The current will spike quite high, almost twice that of the running current when the motors are stopped and started again. Thus any change in the motor requires more initial amperage. This is normally not a problem as the spike will only last for a few seconds, however, in Einstein's case, this could happen rapidly as he changes from one state to another (i.e. turns left, moves forward, moves backwards, turns right). These changes cause multiple continuous spikes that must be accounted for. The power system initially had the system running on one 12V regulator. It did perform for awhile, but it could not handle the continuous changing current, thus we switched to a more robust system. For simplicity, we gave each motor controller one +12V regulator, and ran Einstein's Eyes on its own regulator (to prevent noise interference in the digital logic board). Each regulator had input and output capacitor filters as shown in the schematic below (note: there is a red switch that is between the battery and supply which is fused-there will be a schematic that shows this later):

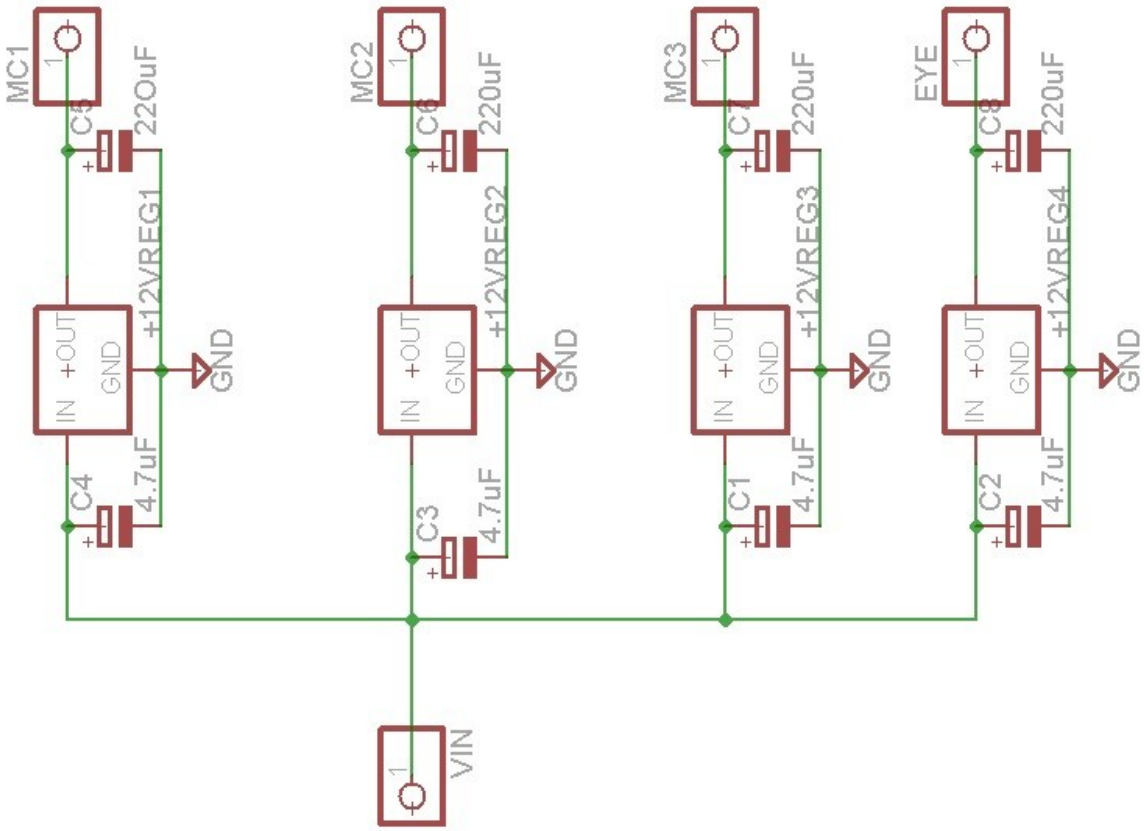


Figure 4: Motor Controller and Einstein Eye's power supply

V_{in} represents the voltage input from the battery, and MC stands for Motor Controller.

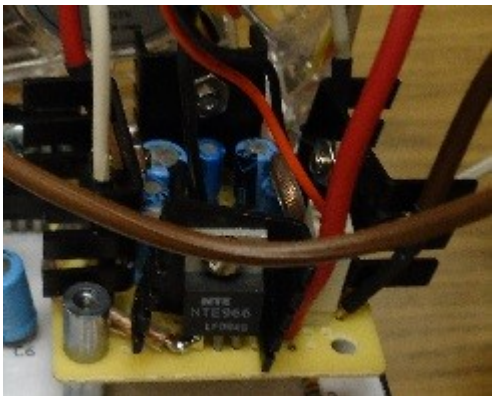


Figure 5: Motor Controller Power Board

Servo System

Originally, Einstein had two mini-SSC II servo controllers where each worked in parallel, one was set up for channels 0-7 and the other for channels 8-15. Also quick notes here is that Einstein's original servo lines ran through a pipe and were difficult to trace. Nathen cut each servo line and reinstalled new connectors for each individual servo.

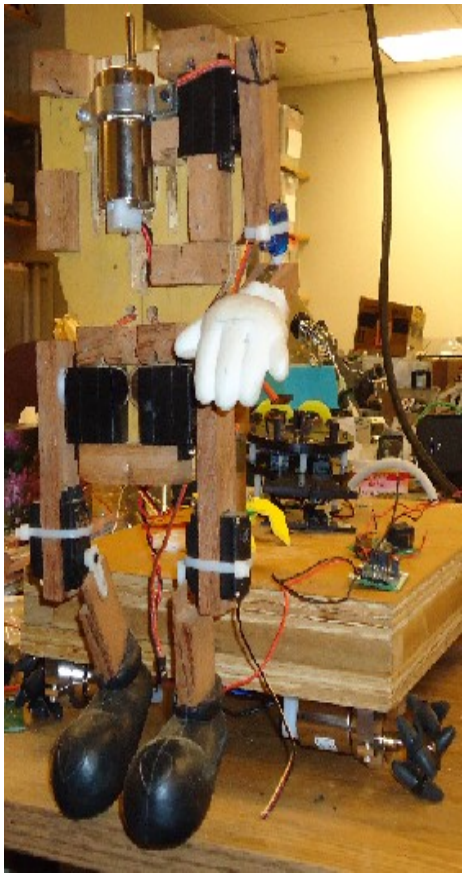


Figure 6: Einstein almost at Ground Zero

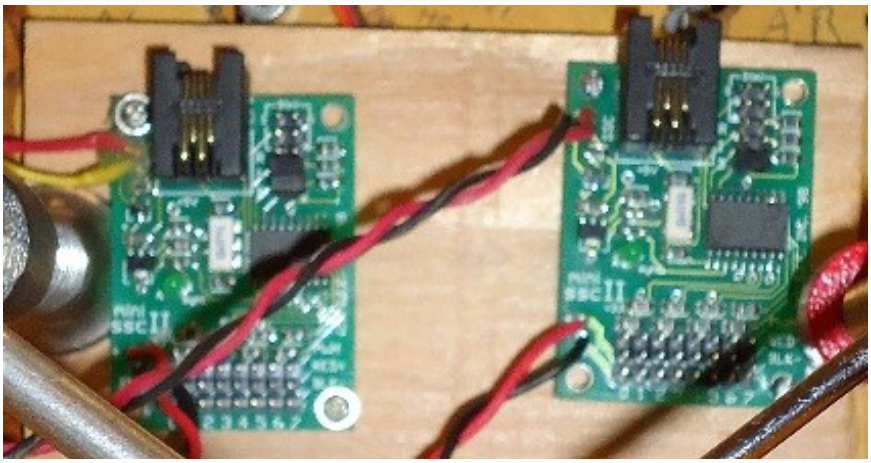


Figure 7: Mini SSCIIs

After Einstein was completed and the power system was built, we were unable to communicate to the mini-ssc Iis using a USB to serial conversion. We were using a program called VSA (Visual Show Automation) which can be found at the following web site: http://www.brookshiresoftware.com/vsa_downloads.htm .

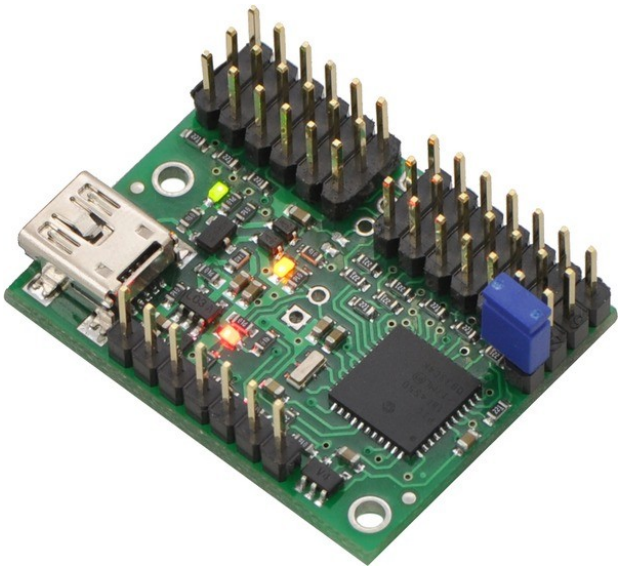


Figure 8: Pololu (18 channel)

We decided to replace both boards with a Pololu 18 channel servo controller (http://www.pololu.com/catalog/product/1354).

This controller was purchased using RAS funding and cost about $46.00 (with shipping). The benefit with this controller was that we could connect directly using USB without a serial converter and we could download a free program to run the servos called Maestro Control Center. The software contained a .dll file that we later used in our program to communicate to the servos. We used the same number variables as the original program, sent the data into the .dll file, then it sent the data bits to the Pololu.

What is important to note here is that all of the servos are on one main buss for the power system. Thus all 18 servos have the same junction to the servo power supply. The servo power system designed on Einstein takes the battery power directly and converts it into 6Vdc. This was where the problems began. We originally attempted to use a single voltage regulator. We did not take into account

that the servos would pull a lot of amperes and due to our inexperience we did not check the amperage rating of the servos. Thus, while testing, we quickly discovered that a single regulator could not handle the current necessary to supply the servos.

One way we discovered to get around this, while still using the regulators, was placing three of them in parallel in order to distribute the current, thus, providing the servos the current needed to function. We used Google to find some help and a particular forum became important. Dean Huster of Electronics Curmudgeon suggested to use small resistors in parallel in order to prevent the regulators from fighting each other (Huster). With this knowledge, we attempted to design a new circuit to handle the current, which resulted in the following schematic:

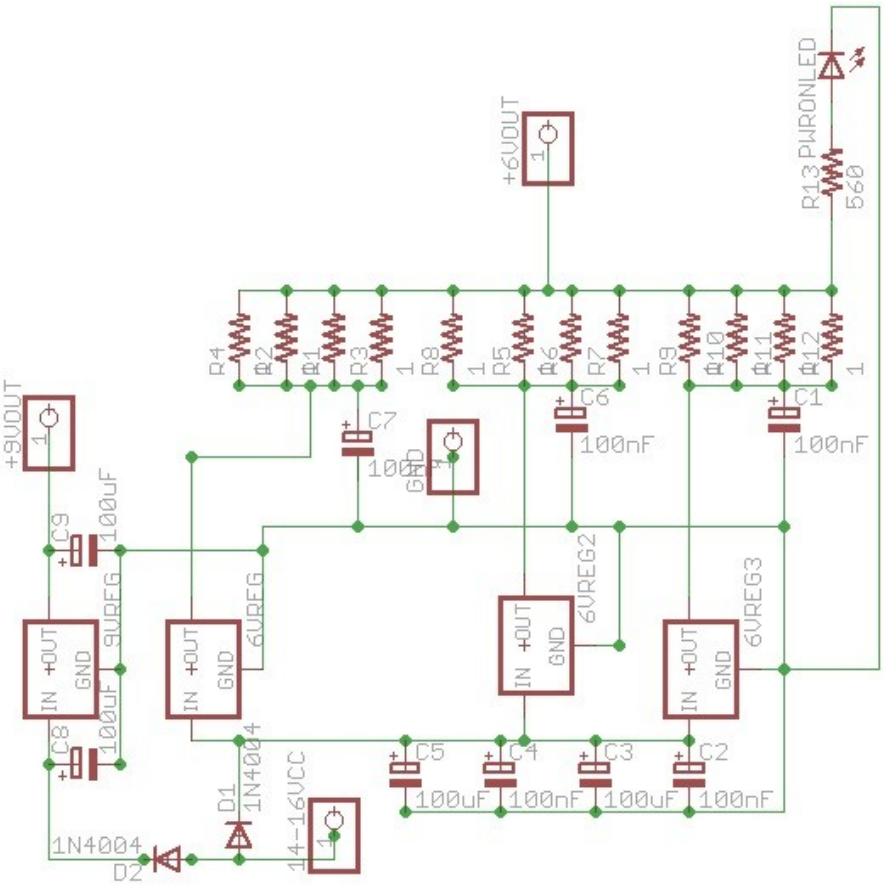


Figure 9: Servo Power Schematic

As shown in the schematic, there are four 1Ω resistors in parallel on each regulator to prevent one voltage regulator taking all of the current over the others. This schematic was then etched by Nathen using the Photoresist method inspired by Make Magazine's Collin Cunningham (http://www.jameco.com/jameco/pressroom/makeoneetch.html?CID=bbmarch2912942&sp_rid=MTgyNDYxNTA4MzgS1&sp_mid=2912942). Nathen decided not to etch the motor controller power supply system due to the simplicity of the board. Nathen also used his own money for all of the etching chemicals, supplies, and electrical components, to preserve more of the RAS funding. Most supplies were purchased from Jameco Electronics online.

Nathen first used a program called, Eagle Layout Editor (version 5.11.0) from cadsoft to draw out the trace lines to etch the board. Tin leaned that the Eagle Layout Editor is a valuable asset to drawing circuit traces, and developing schematics, however, it is difficult to find the correct components in the library. Each individual component has its own file within the library, but some manufactures had contributed and added numerous other components as well. Also, some of the components are imbedded in other libraries. For example, the resistor library also has capacitors as a sub-library and there is already a capacitor library by itself. We were able, however, to add a component in the library because it did not exist. The component added was the LM2677, which can be found on the DVD. This was actually a handy attribute to the program. Tin discovered that the hard of part etching is ensuring that there weren't any glitches in the trace lines after printing onto the transparencies. We also didn't check the trace lines before we etched the buck converter the first time, and discovered errors in the traces after it was too late. Thus the lesson learned was to ensure that we have thoroughly examined the traces. Figure 10, shows the eagle trace lines of the 6Vdc voltage regulator power conversion for the servos.

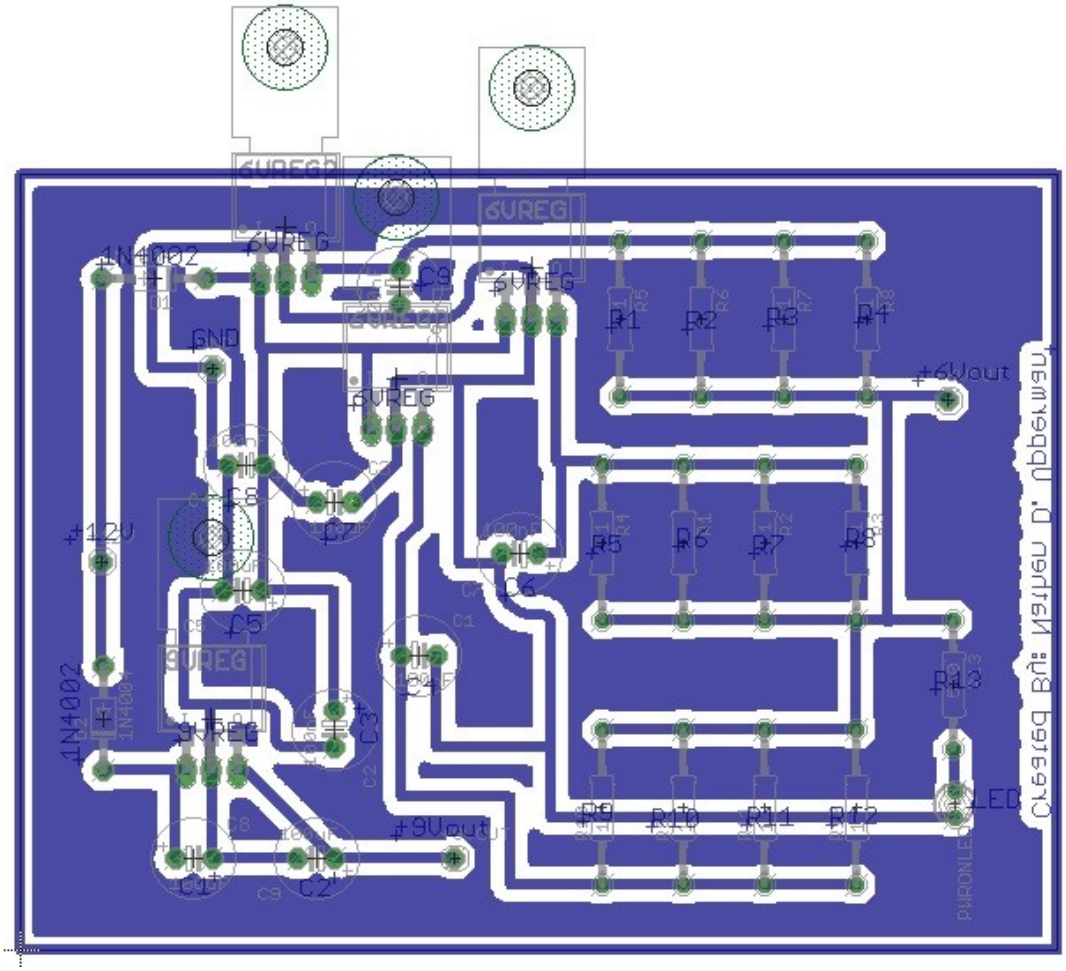


Figure 10: Eagle Schematic

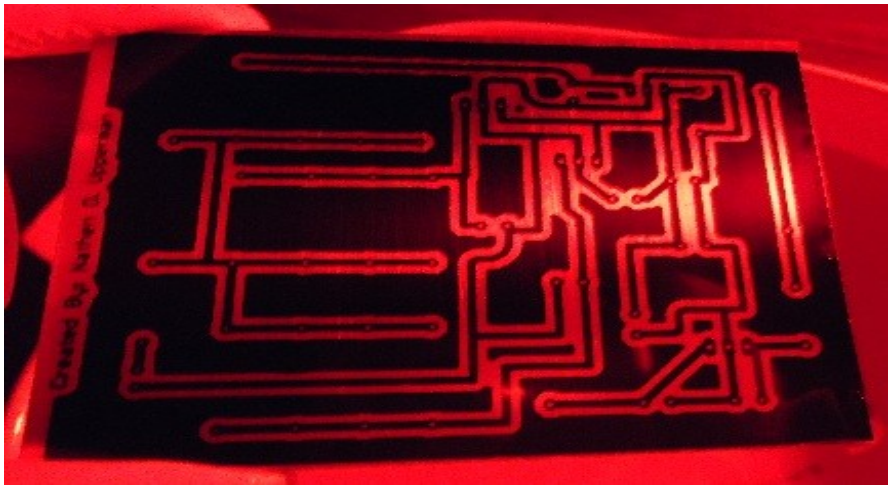


Figure 11: Etching Servo Power Board

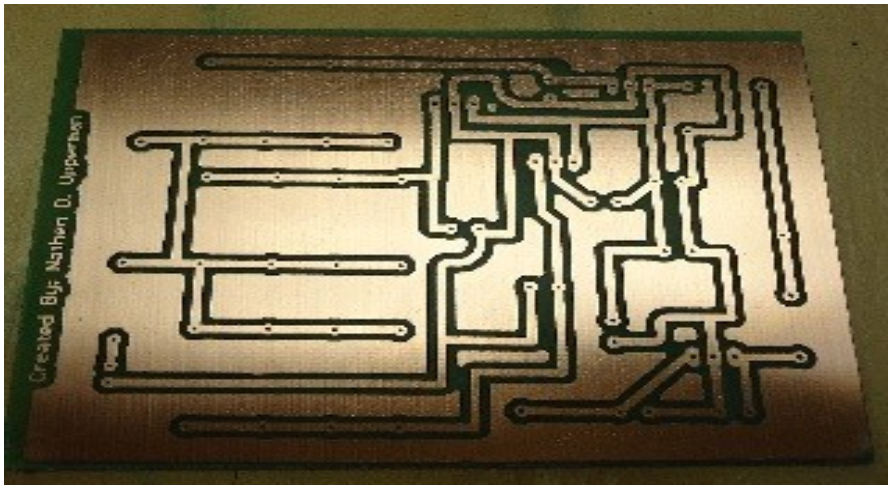


Figure 12: Completed Etching Process of the servo Power Board

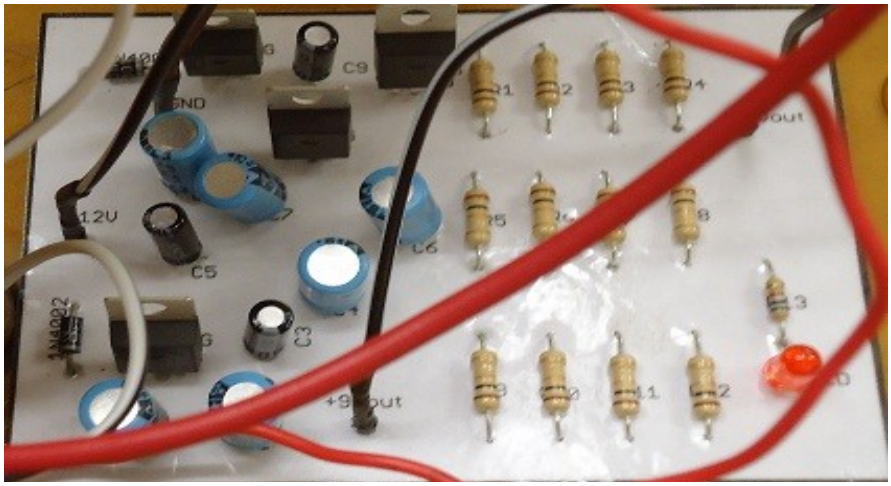


Figure 13: Completed Servo Power Board

The board shown in Figure 13 was slightly modified to stack the motor controller power supply on top so as to take up less space and a fan was added to cool the regulators as they get quite hot.

Figure 14: Motor Controller and servo board together

Note that Figure 14 shows a +9V output that is not being used. This is because Nathen originally designed this board for the mini sscII boards that needed an external power supply. We didn't run into the power problem then, since we could not communicate to the mini sscIIs to activate the servos. The Pololu that replaced the mini sscIIs utilizes the USB power for the controller, thus not needing an external power supply. However, if anyone later decides to utilize a microcontroller to eliminate the need of the USB cable, the Pololu would need external power, which can still be used by this 9V-output voltage. It runs off its own separate regulator connected directly to the battery supply, thus the power system would be independent of the rest of the circuit, with the exception of the input power. This circuit still did fail from time to time during a high servo load. We tweaked our program to where each servo was not pulling its full current by slowing the servos acceleration and speed. There were many problems with the voltage regulators themselves as mentioned below:

- Not very efficient power supply (drops the voltage by dissipating excess energy-heat dissipation)
- Wastes lots of battery power
- Have low current limitations (1-2 amps max)

Despite the drawbacks of the voltage regulators, they are the cheaper and less complicated solutions to power problems. These components also have a heat cutoff point where if the component reaches a certain temperature they will shut off, which occurred from time to time. Since Einstein is pulling much more current than a typical regulator can handle, and high efficiency loss due to the amount of voltage

being stepped down, we decided to look into alternative methods. The buck converter (a fast switching regulator-nonlinear versus a voltage regulator which is linear) came up as the best solution and according to Altera, “The clear advantage of switching regulators is efficiency, as minimal power is dissipated in the power path (FET switches) when the output supply voltage is sufficient for the load state” (Altera 1). Figure 15 shows a typical topology for a buck converter, and in this case, was the one we tried.

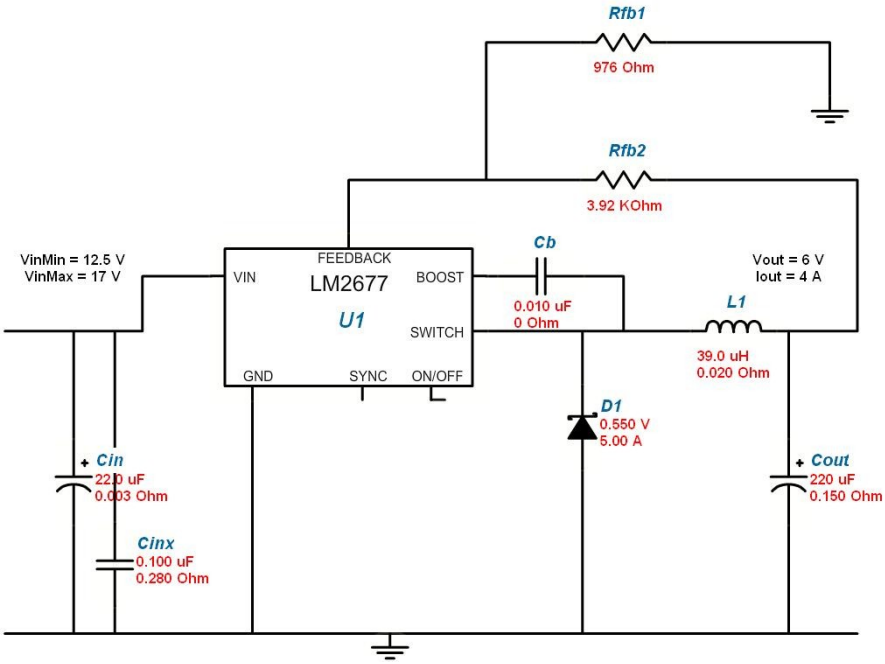


Figure 15: Buck Converter

Source: National Semiconductor

We could have purchased the board already completely built, but it would have cost more than building it separately. On the other hand, building the board leaves room for errors in the schematic and time. Nathen made the decision to build the board, since he was paying for it. Once we obtained the parts that Nathen had purchased (about $20.00 from Jameco Electronics), we tested the buck converter on a protoboard. The first time, it failed as the schottky diode was installed incorrectly forcing all of the current to flow directly into ground. Nathen discovered this and corrected it. The second test succeeded with a 6.02VDC output. We did not load test the circuit, which we should have, as we will later see. The buck converter was drawn in Eagle by Nathen and Tin and etched as shown in figures 16 and 17.

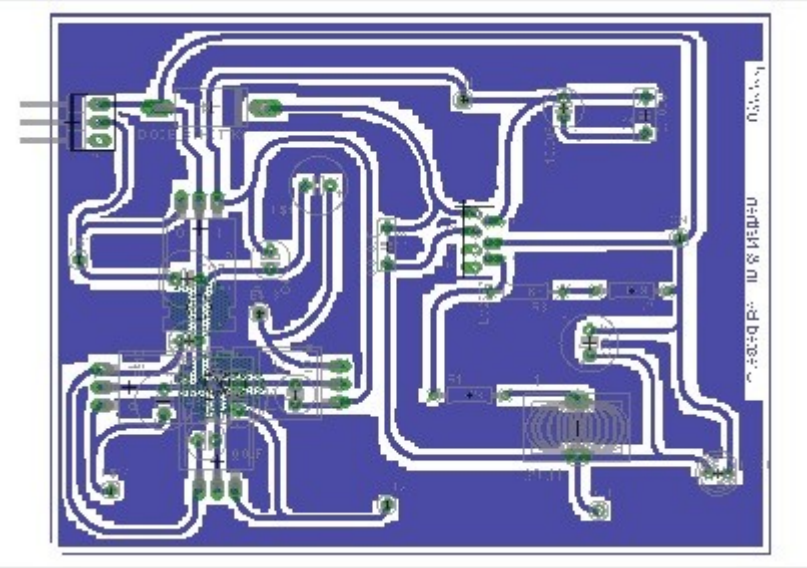


Figure 16: Buck Converter drawn in Eagle

Figure 17: Buck Converter Etched

The etched board was then installed on Einstein and powered up. The system seemed stable and without any power issues on a minimum load. Once Einstein's servos were engaged, the LM 2677 became super hot and began to fail. According to the datasheet, this particular component should have been able to handle 5.0A , and provide about 92% efficiency with a maximum input voltage of 45V. There is a possibility that the chip was damaged when the diode is inserted improperly during the testing phase, however, very unlikely, as it was still able to provide the initial 6.02V without a load. It seems this component failed to live up to the manufacturer's specifications. Thus we ended up keeping the voltage regulation power system as time and funding got low. Even though Nathen used his own funding, frustrated that the board didn't work, and concerned about the waste of cash, he learned a lot from the experience and new troubleshooting techniques.

Through this experience, Nathen believed his knowledge in Electrical Engineering was greatly elevated over this entire power system process and had learned a great deal of how to design a new and better power system. Even though it would have been simpler to have just used multiple batteries, Nathen would not have gained such a grasp of circuit design of DC-DC conversions and was well worth the trouble, especially since it will come up again in his future career. As of the writing of this paper, he will continue to explore the buck converter and perform a much more thorough analysis on the system that he could not do before due to the time constraint, see if he can find any other reason the buck failed, and test other components to find a better i.c. that can handle the amperage. It would be better to find an i.c. that can do the job rather than building it from scratch, since the i.c. would already have the pulse width modulator already installed and would automatically change its duty cycle and feedback lines as the load increases or decreases. The next page will show two schematics that have combined multiple sections if Einstein's power system.

Full Power Schematics

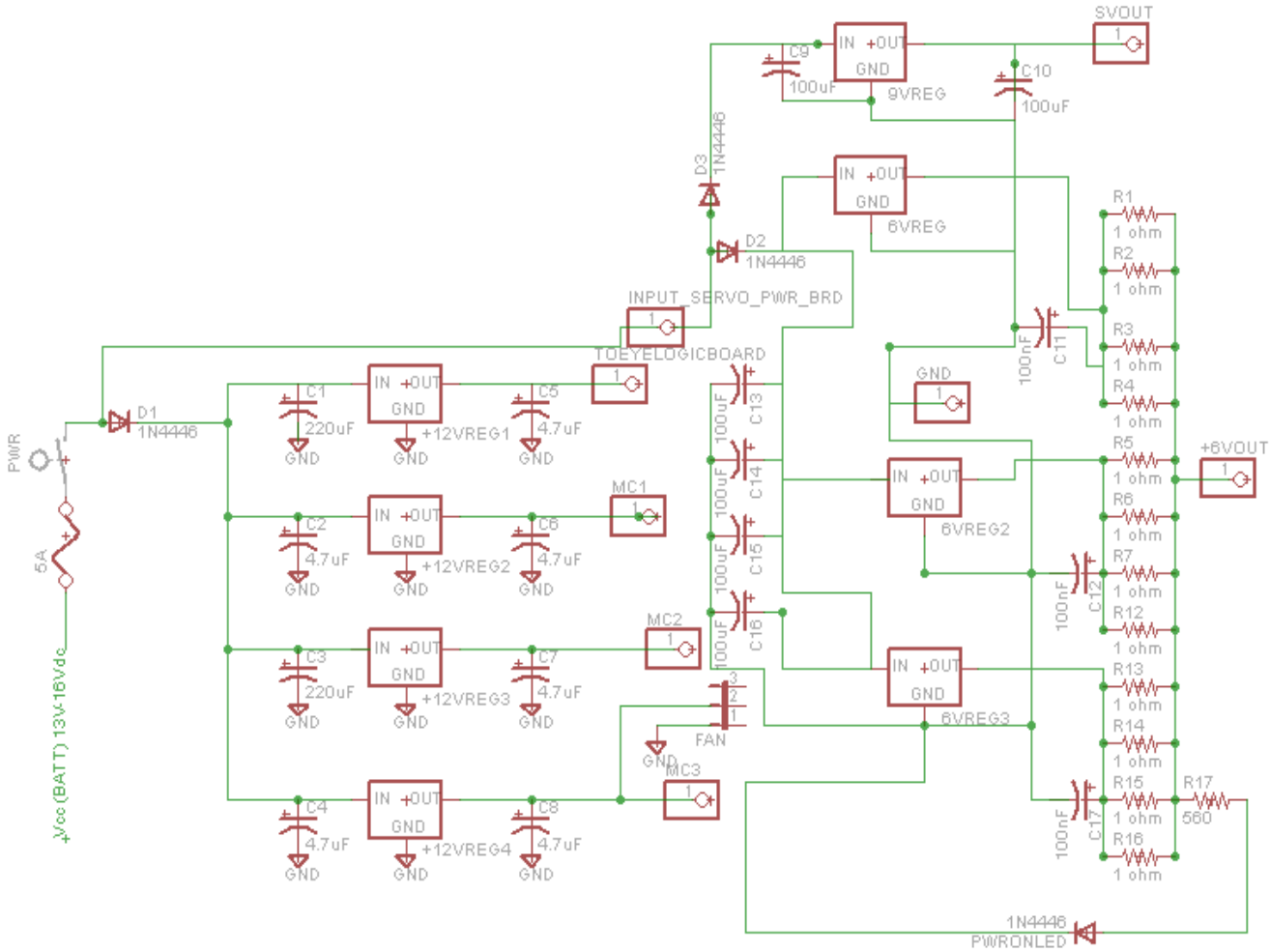


Figure 18: Merged Servo and Motor Controller Power System

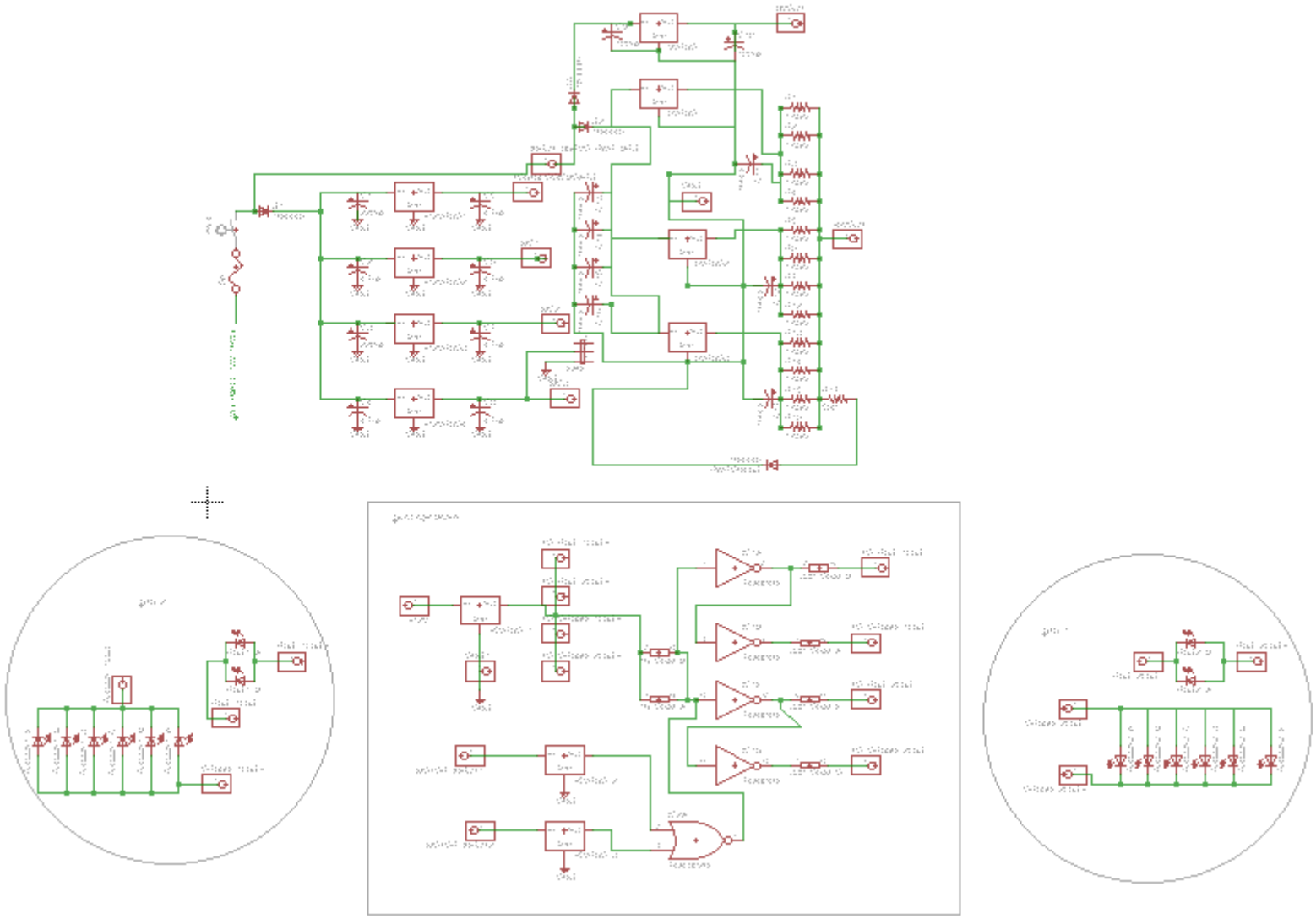


Figure 19: Full Power Schematic

Einstein's Eyes

Originally, Einstein's eyes were quite plain with no real physical application as shown in figure 20.

Figure 20: Einstein's Original Eyes

Nathen came up with an idea to build eyes using LEDs, but they would change color if some event occurred. He decided to design a logic circuit similar to a logic probe, but he wasn't sure how to trigger his eyes when some event occurred. He decided to build it based on Einstein's Neck rotation, thus if the motor moved to the left or right, his eyes would change from a yellowish state to a red state. To do this, he used a truth table shown below:

Motor 1	Motor 2	Output
0	0	0
0	1	1
1	0	1
1	1	X

X= Don't Care

Table 1: Einstein's Eyes Truth Table

He then took advantage of the don't care condition and turned the output into an OR gate. Since he didn't have an OR gate on hand, he used his NOR gate and inverted the result with a Hex

inverter to get to the equivalent output. This took care of the logic and the rest was the result of running a few hex inversions together, to create an inverse of what the other led was doing.

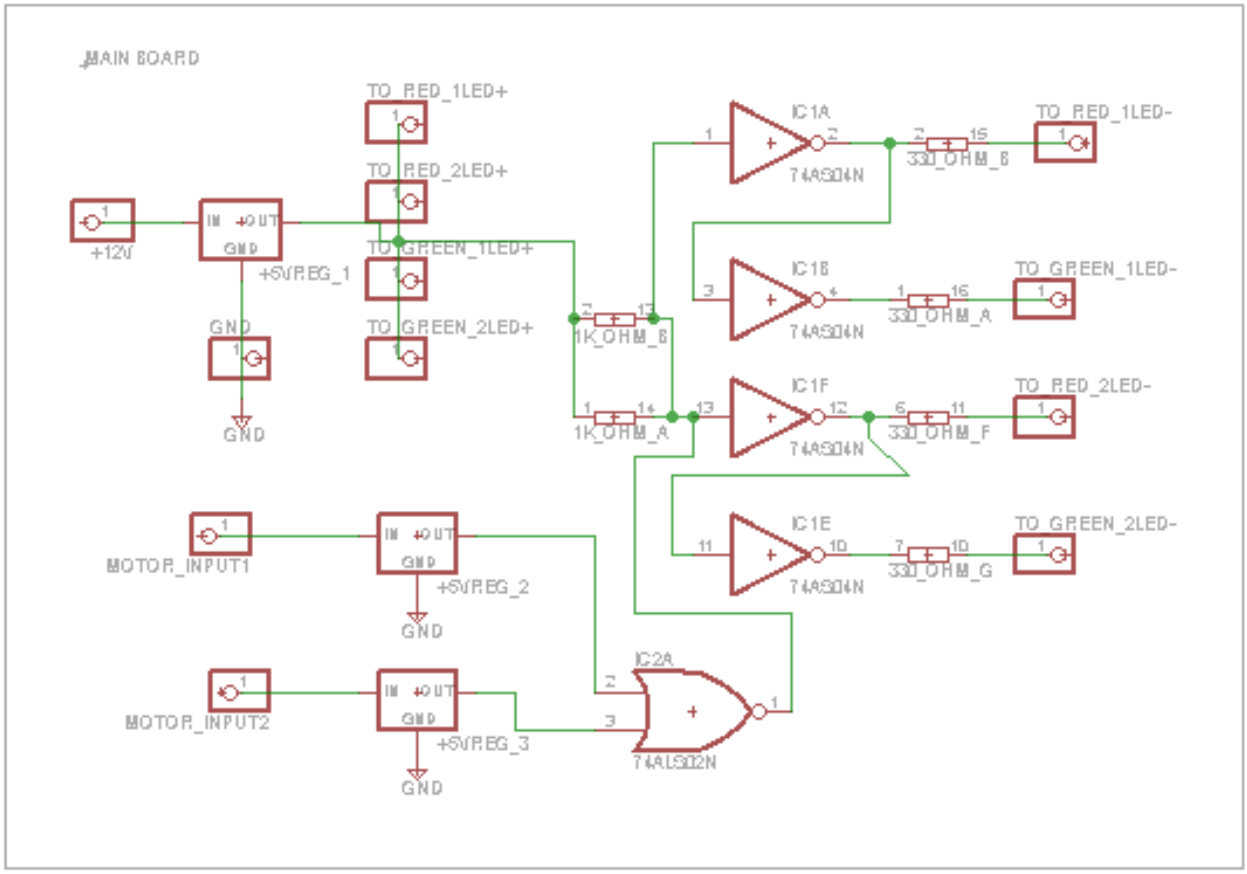


Figure 21: Einstein's Eyes Logic Schematic

Then Nathen created the eyes, which were designed to be circular, as shown in figure 22.

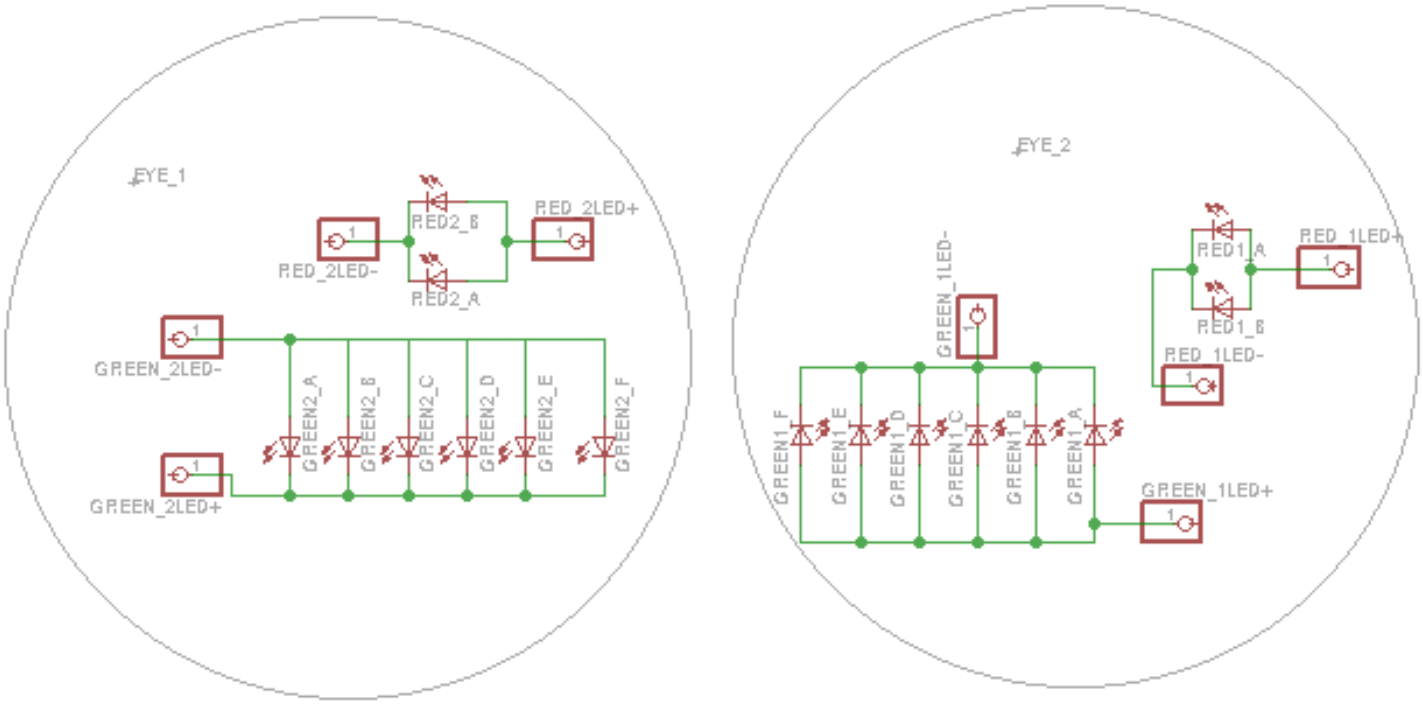


Figure 22: Einstein's Eyes

After developing the schematics the system logic was completely designed on a breadboard and tested, where the eyes have been built on trimmed up boards and the rest of the logic was on the main board. Nathen originally did not know what to do with the eyes or what size to make them. He discovered that certain water bottle caps were clearer than others, which provided a great way to hide the LEDs enough to where they are difficult to see, yet allow the light to shine through. A bonus effect was that the caps also refracted the light enough to where it merged the LEDs lighting up the cap really well. The ones on currently on Einstein were a little to clear to Nathen's liking, but still worked. There were other caps discovered later that had a much better effect.

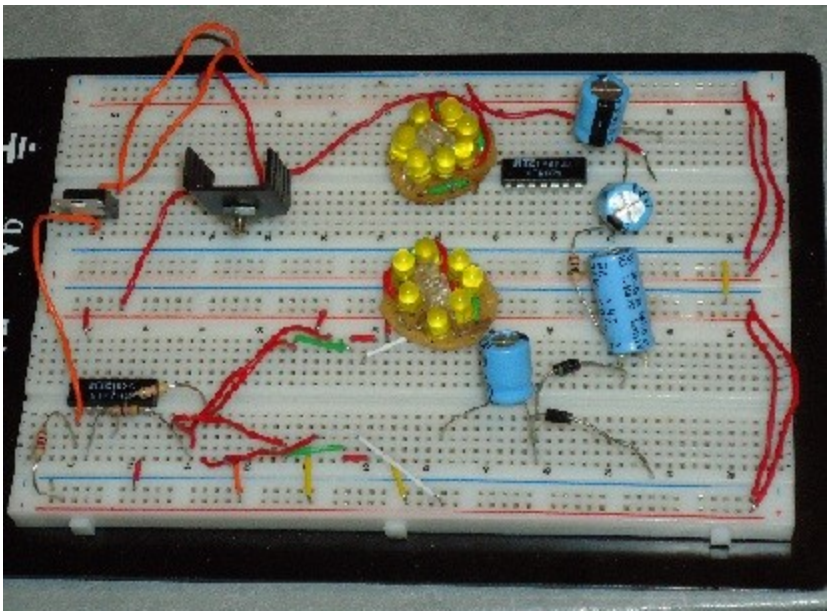


Figure 23: Testing Einstein's Eyes

After the testing phase, Nathen etched the main board that controlled the digital logic. In the schematic in figure 19, note that there were two +5V voltage regulators added. This was due to the input of the +12V motors. The logic gates can only handle about 5Vdc, thus, the 12V power needed to be stepped down. Since only small amount of voltage is required to change the logic, it will not effect the motor's performance. Later on, these inputs could be placed on a microcontroller to control Einstein's eyes based on what is programmed in the controller. As long as there is about 3V or so on that line, his eyes should change. Further experimentation may be required since the 12V regulators will take some voltage from the line even though a higher voltage is not implemented into the system.

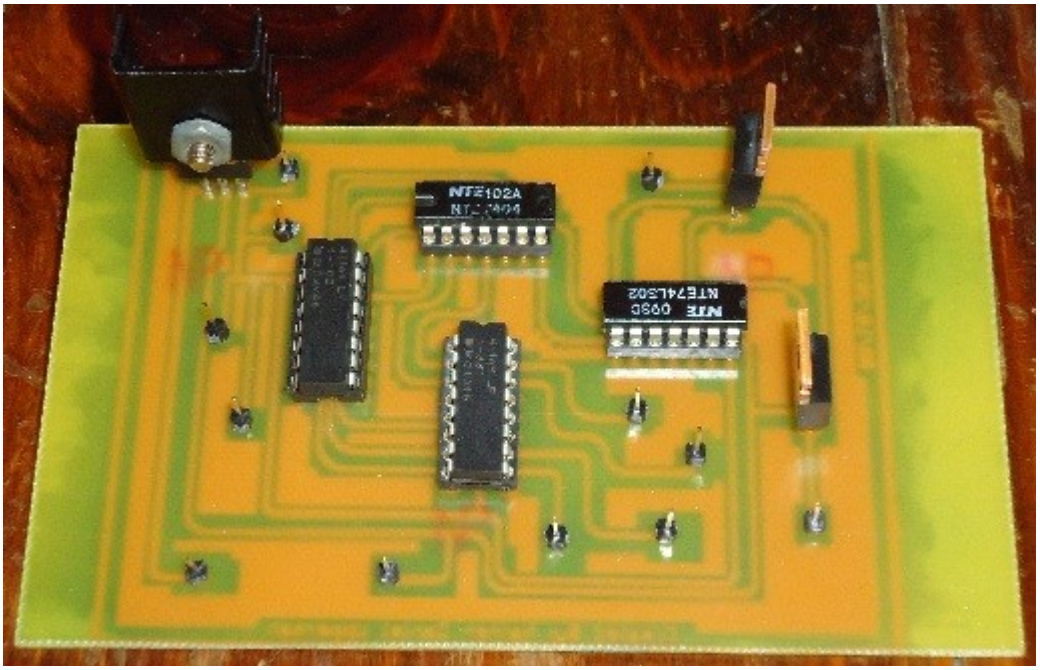


Figure 24: Logic Board

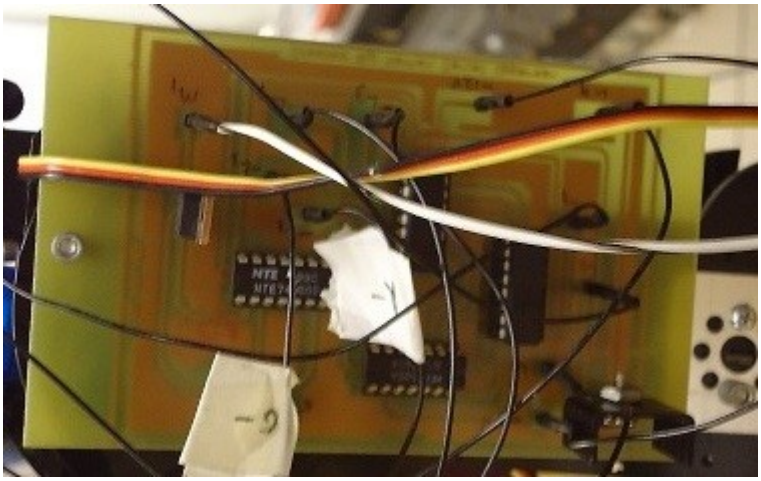


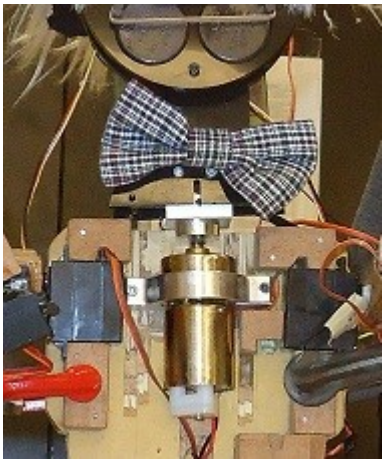
Figure 25: Board Installed

Figure 26: Eyes Yellow

Figure 27 : Eyes Red

Einstein's Neck Rotation

Einstein originally had a servo attached to his neck. This servo was removed and one of the 12VDC motors was installed. This was done since the servo could not support the weight of Einstein's head and resulted in sloppy uneven turns. We do not have an image of the original neck, but the new neck was completely redesigned by Nathen to support the head. A metal bracket was installed to add more support thus providing a more stable neck. The problem later was that since the motor was not stepped, it could not be controlled very well.



The motor would not stop in the same location every time his neck was activated. Since there are wires connected to the back of the head they would eventually become wrapped around the neck. This problem was never solved. The program currently will provide only enough power to the motor to trigger the logic in the board and not rotate the neck.

Figure 28: The Neck

Einstein's Wheels

Einstein's base wheels were designed by Kjersten using the mecanum wheels. When my group started using the 12V motors, our wheels kept falling off when Einstein turned. We discovered that the wheel hubs were made out of plastic and the points where a few of the keys were supposed to lock the hubs in place were cracked. We tried replacing the hubs with the Tetrix aluminum ones, however the holes did not match the wheel mount holes and there was no way to drill in new holes and tap in threads with the limited distance on the aluminum hub. The 12Vdc motors also would not turn fast enough to allow the wheels to slip where we could take advantage of the side to side motion. We believe this was due to the weight of the base, since there are three solid pieces of plywood stuck together. We replaced the wheels temporarily with ones that came with the Tetrix kit, which also had aluminum hubs. We liked the performance of the wheels, so we kept the wheels on Einstein, although they limited the base mobility.

Figure 29: Mecanum Wheels

Figure 30: Tetrix Wheels

Einstein's NXT Implementation

When the Einstein Project started, he was designed for the implementation of Lego's NXT.

Figure 31: The NXT

The NXT requires software that must be purchased, such as RobotC, which is what we had used. The NXT has four sensor ports (1-4) and three motor ports (A-C). We used one of the sensor ports to connect to the first motor controller (port 1) and the other two motor controllers were daisy-chained off the first one. The other three ports were used as sonar sensors (port 4=front, port 3= right, port 2 =left).

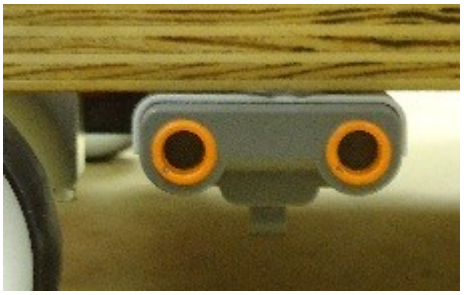


Figure 32: Sonar Sensor

The sonar sensors on Einstein were originally installed so that Einstein could run a program on the NXT and travel through a labyrinth, however, we soon discovered that these sensors were not reliable enough as Einstein moved. We were undetermined as to why exactly. It could have been a poor sensor design by Lego, or the NXT itself. Regardless, we still tried to use them to prevent Einstein from turning into a direction that would cause him to crash into a wall if the camera detected a face and wanted Einstein to turn into that direction.

Nathen had written a basic program in RobotC (see Nathen's Original Code for the NXT) that ran the base motors and sonar sensors without any other implementation. The program worked, but needed a lot of work. Gehad took the programming from here and slowly manipulated Nathen's original code. Soon he had added bluetooth code since the NXT has Bluetooth technology already installed and utilized his cell phone to control the NXT thus controlling Einstein wirelessly. Gehad also wrote code to use Nathen's laptop to control Einstein wirelessly, which the report has more details of in the software section. Due to the limitations of the code, we recommend replacing the NXT with a microcontroller, such as the Arduino, where the software is open-source. It may also provide faster processing, better memory handling, and perhaps the sonar sensors will work better, utilizing interrupts.

Figure 33: Arduino UNO

We used a USB Logitech camera to perform our live video capture. We discovered that some cameras are not compatible with the new 64 bit Windows 7 operating system and we also discovered that the camera's attributes such as automatic light correction caused glitches in our OpenCV tracking for image and face detection.

Figure 34: Logitech camera

Phone Cable to NXT Cable Conversion

We had a lot of problems with the short wires that we had purchased from the Lego Mindstorm website. Nathen did some research online and found that there was a way to convert a phone cable to the NXT cable (Gasperi-Lego Mindstorms NXT: Synthesize-your-own-NXT-Cable). The web site was helpful, but Nathen discovered some issues and formulated a different way to do it. The phone cable has 3 conditions that must be addressed to convert them to an NXT cable:

1. The tabs must be removed and offset
2. The cable must have 6 pins
3. The cable must be untwisted

Nathen had used an Exacto knife to cut off the original tab. He found that this was best since the cut didn't remove much of the plastic, which is needed to remount it. Then the tab needed to be ground a bit on each side as it was to wide to fit into the tab placement of the connectors. He found that sizing the connectors took a little bit of grinding and placement. Then, he used hot glue to mount the tab on the far right corner offsetting the tab. Nathen suggests to use a different type of glue that is designed for gluing two plastic pieces together, such as a polymer glue, since the hot glue occasionally came loose.

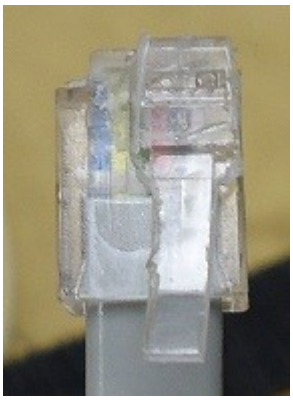


Figure 35: Modified Tab

Figure 36: Original phone Cable

Figure 37: Actual NXT Cable

The pins on the connectors must have 6 pins. When Nathen was building the new cables, he realized that the phone cables have different pins. Also, the phone cables are twisted pair. You have to untwist the wires. If you happen to have a crimper, you could cut the cable and put on a new crimped tip. Nathen didn't have one and due to their expense he instead cut the cable and manually untwisted the wires and re-soldered them. It is important to note here that it is difficult to solder these wires as some of them can be polarized wires where they will not absorb the solder and it was difficult to remove the insulation from the small wires without accidentally cutting the whole wire in half.

Figure 38: Untwisted Phone Cables

Einstein's Body Change

Einstein's Legs needed to be fixed. The original designer did a poor job at connecting the legs to the servos where his legs were flimsy and rigid. Nathen rebuilt the legs and sank in the disks to the servos, where the connections were solid.

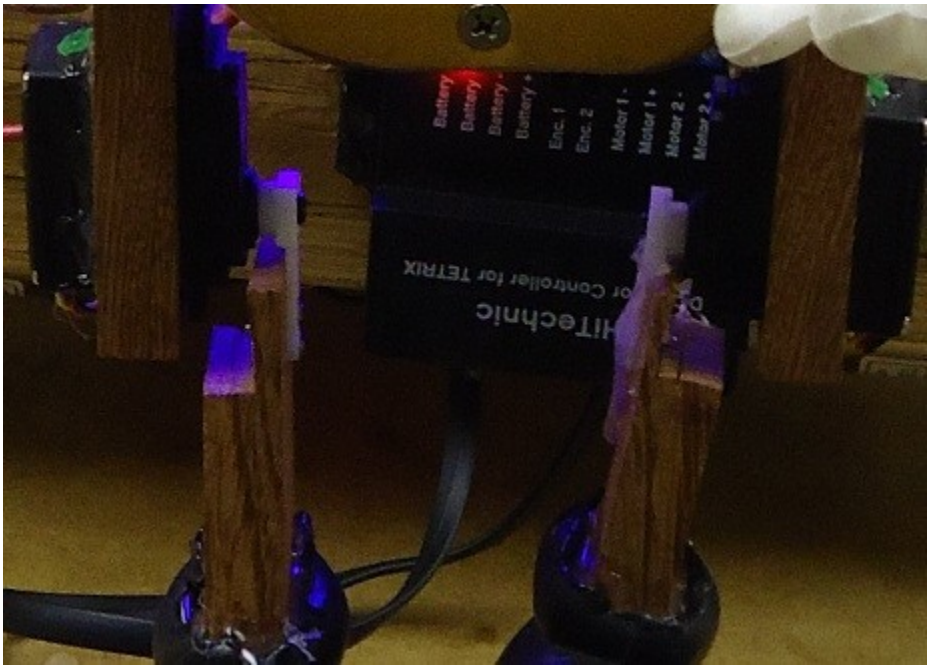


Figure 39: Original Legs

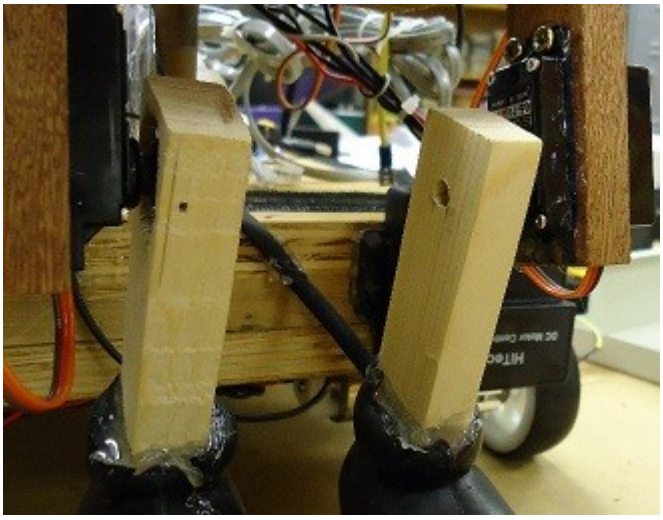
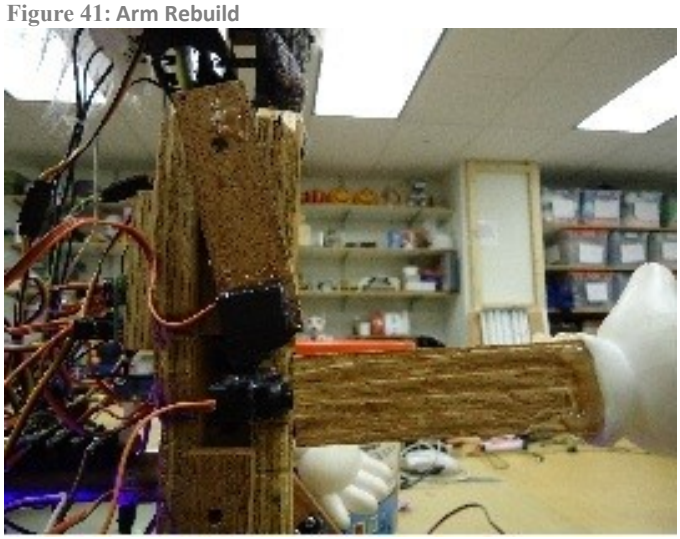


Figure 40: New Legs

Nathen also rebuilt the arms adding servos to give Einstein more mobility.

Figure 41: Arm Rebuild



Here are a few images of the bottom of Einstein's base, where the motor controllers are daisy chained and you can also see that two motors are connected to each motor controller. Nathen had set up the original NXT program so as to account for the motors positioned on opposite sides when trying to move forward or backwards. What would be considered forward for one motor, could be considered backwards for another motor since they are on opposite sides of the base.

Figure 42: Base Bottom 1

Figure 43: Base Bottom 2

Software Components

OpenCV build & configuration

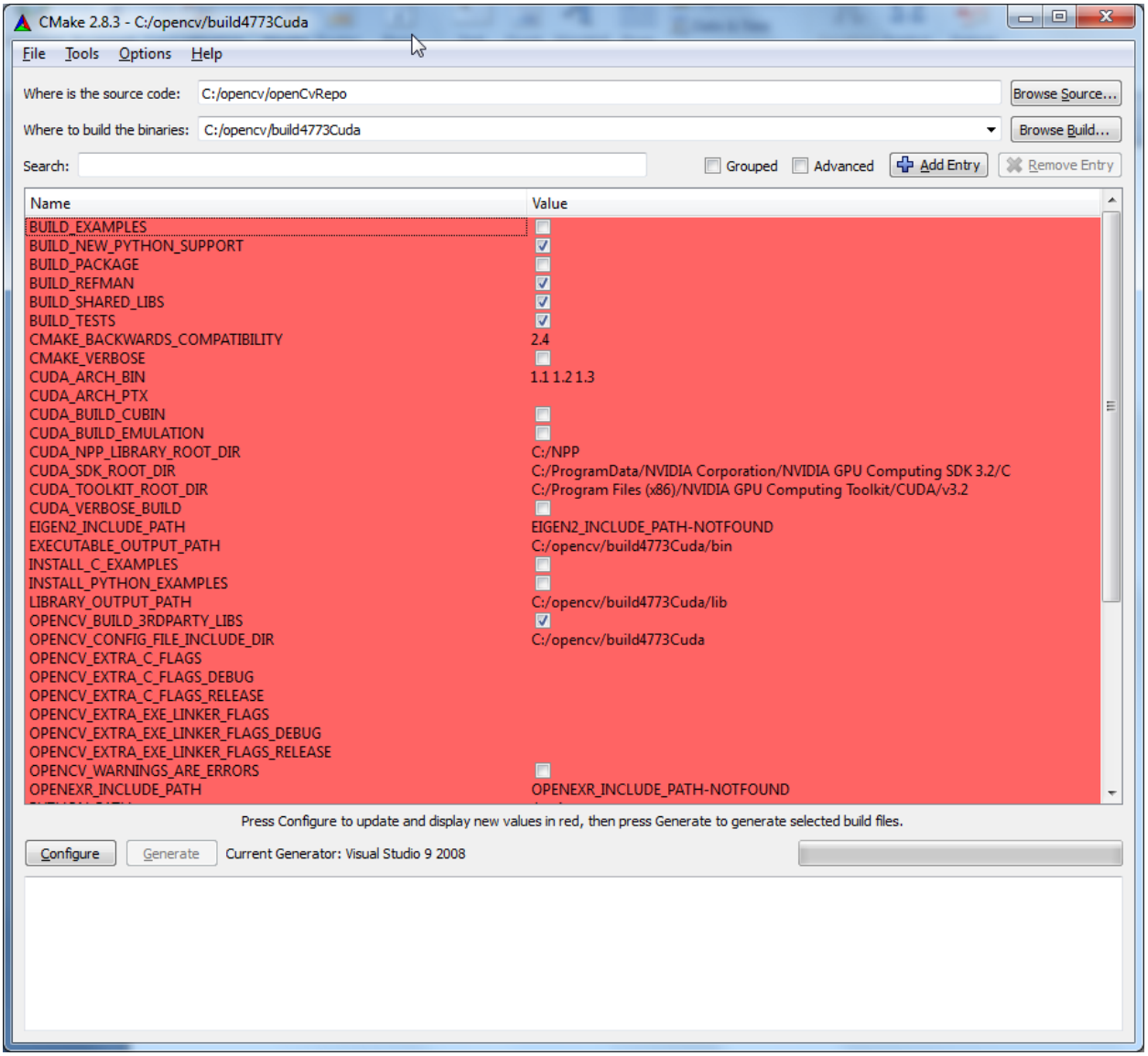
How to build OpenCV

Gehad decided to build the latest version of OpenCV from the repository instead of the pre-built OpenCV to enable us to design custom builds of OpenCV to add features to it like CUDA support and TBB. The process required:

- CMake 2.8.3
- SVN client
- Visual Studio 2008

The process:

- First download a working copy from the repository: https://code.ros.org/svn/OpenCV/trunk/OpenCV/
- Then open CMake and choose the directory where the program was downloaded and the place it where you want to build the OpenCV library.
- Click the configure option and choose the compiler, then click finish (VS 2008 in our case)
- Choose the desired OpenCV configuration, addition, and where the library and binary files should be built
- Click configure until no red lines are shown
- Then click Generate; this will generate a solution file where it was specified
- Go to that directory and open the solution file with VS2008
- Right click on ALL BUILD project then click build
- It will take around 10 minutes to complete-maybe more if adding other components like TBB or CUDA
- After it has finished, right click on INSTALL and click build



Evironmental Variables

For compatibility, Gehad suggested having the same environmental variables across all the machines used for development, which will be required in the machine using the solution file. Here is the convention we used

variable name	variable value
OPENCV_LIB_DIR	should be the path to OpenCV library dirctory for example: C:\OpenCV2.2\lib

OPENCV_INC_DIR	should be the path to include directory for example: C:\OpenCV\include\
OPENCV_BIN_DIR	should be the path to bin directory C:\OpenCV\bin\

Human Part Detection

In OpenCV we have the following haar classifier that is trained to a specific detection.

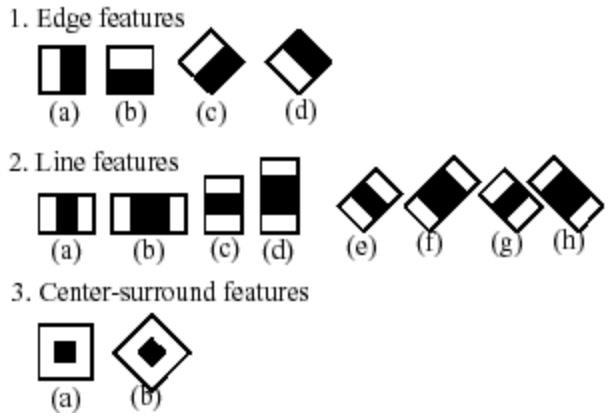
- haarcascade_eye.xml
- haarcascade_mcs_eyepair_small.xml
- haarcascade_eye_tree_eyeglasses.xml
- haarcascade_mcs_lefteye.xml
- haarcascade_frontalface_alt.xml
- haarcascade_mcs_mouth.xml
- haarcascade_frontalface_alt_tree.xml
- haarcascade_mcs_nose.xml
- haarcascade_frontalface_alt2.xml
- haarcascade_mcs_righteye.xml
- haarcascade_frontalface_default.xml
- haarcascade_mcs_upperbody.xml
- haarcascade_fullbody.xml
- haarcascade_profileface.xml
- haarcascade_lefteye_2splits.xml
- haarcascade_righteye_2splits.xml
- haarcascade_lowerbody.xml
- haarcascade_upperbody.xml

- haarcascade_mcs_eyepair_big.xml

Each one of the classifiers can be used to detect a feature that the names of the files specify. These are trained by Intel using positive images, false images, and a combination of false and positive images library. In our case these pre-trained cascades worked well for us, and we didn't need to train a new classifier.

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers. The feature used in a particular classifier is specified by its shape, position within the region of interest, and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied)."

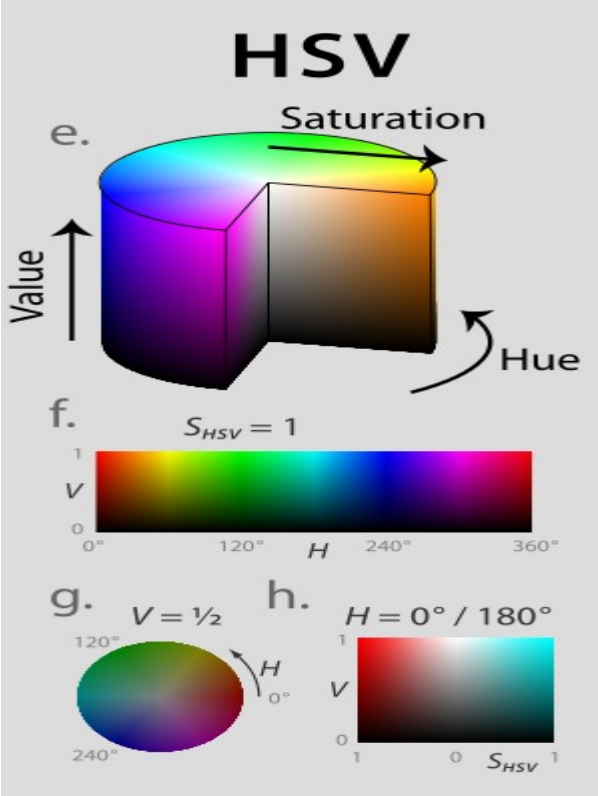
The current algorithm in OpenCV uses the following Haar-like features:

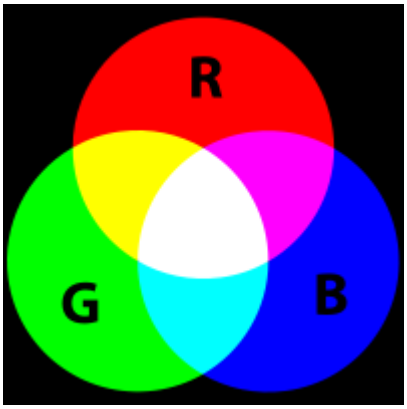


These features are used basically to detect any object and are used upon analysis of the classifier that is pretrained. Check Development Process Summary for more detail.

Color Detection

First, we convert the image from RGB(Red Green Blue) space to HSV (Hue Saturation Value)





where RGB consist of three layers Red, Green, and Blue when added together creates one color, however, in HSV space it's three laters Hue, Sturation, and Value. We can see from the illustration above how can we pick the color we want easily from HSV space and it would be difficult to do so in RGB space.

Then, we pick a range of each of the layers and eliminate pixels that are out of the range. We will end up with an image with 0 in the area that's out of range and 1 in the area in our range of color.

Check Development Process Summary for more detail.

Development Process Summary

Overall the project went through three phases: OpenCV learning, design discussions, and finally software and power systems development. Our group's approach was to first get familiar with the existing system and potentially useful software, develop test modules, and finally integrate them into a demo.

During the first phase Ben and Gehad focused on getting familiar with OpenCV which was parallel with the class homework assignments on image manipulation through morphology. Nathen focused on RobotC NXT programs to test motor control and head/neck functions. At that time our group decided on using two different tracking modes, human tracking which Gehad would focus on, and object tracking which Ben would focus on. Based on some example code found at "http://www.aishack.in/2010/07/tracking-colored-objects-in-OpenCV/", Ben worked on modifying the code to track yellow objects, which worked great at home under dark lighting conditions. The approach was to convert an image from RGB to HSV format (specifically "CV_BGR2HSV"), apply minimum and maximum threshold for hue, saturation, and value, and then find the area and center of the remaining pixels. With some tuning this was better than the common RGB format for both ignoring noise and identifying the color over a range of intensities. Unfortunately the color yellow was more susceptible to getting washed-out in typical fluorescent lit areas around PSU, so yellow object tracking was abandoned in favor of either bright blue or bright green. We discovered a green dry-erase marker (specifically the cap) was much easier to see under fluorescent lights. The marker's cap had a particular shade of bright green which could be recognized better in changing light conditions and distinguished from darker green objects. Unfortunately if the marker was moving the thresholded image got noisy, but using some erosion improved object identification.

Willow Garage OpenCV documentation for HSV:

http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_ImageProcessing.htm

 # RGB=>HSV (CV_BGR2HSV,CV_RGB2HSV)

V=max(R,G,B)

*S=(V-min(R,G,B))*255/V if V!=0, 0 otherwise*

*(G - B)*60/S, if V=R*

*H= 180+(B - R)*60/S, if V=G*

*240+(R - G)*60/S, if V=B*

if H<0 then H=H+360

The hue values calculated using the above formulae vary from 0° to 360° so they are divided by 2 to fit into 8-bit destination format.

In theory HSV format could be said to be based on a six-color hue, red, green, blue, cyan, yellow, and magenta. In practice, however, we learned by communicating with other robot teams that this format works much better with isolating particular shades of red, green, and blue under varying light conditions than it does with cyan, yellow, and magenta. During this time Gehad explored different face recognition approaches using built-in OpenCV routines and XML classifiers as well as researching available methods to speed up computation using multiple cores, threads, and CUDA (see below for details).

During the next phase Ben focused on coming up with an overall plan to integrate the software and hardware, which we later agreed upon as a group. The details of this are in the document "Robot Brain pseudo code.txt" which is part of the group of development files in RAS's version control. This document lays out a very high level plan for integration, control of servos via predefined behaviors, and a plan for NXT motor control that permits softer stopping and starting of the robot's base. The plan was to have image data scanned with OpenCV functions to recognize a human or brightly colored object; in either case a center coordinate and area in pixels was returned. Based on the center location and area greater than 0, a behavior and magnitude would be selected. The resulting servo behavior was set to correspond the motorized wheels tracking the object thus moving forward, backward, turning right or left, or stopping with a value for speed (and/or torque). For each behavior the servos would go through a corresponding predefined set of positions at a rate proportional to the motor speed. The advantage of the design is that the speed of servo motion could change on-the-fly. The servos could also go through a sequence of positions and stop at the final position, as well as support new behaviors in the future. Our implementation was done through first getting a group of test programs to prove the individual components worked for OpenCV, NXT/Bluetooth, and the servos (Pololu Mini Maestro 18), then integrating them in the last phase. During this time Gehad helped with debugging, spearheaded getting all people developing software to share code through revision control while ensuring library compatibility, and developed Bluetooth test programs in both RobotC and Android compatible language (see below for details). In the end we had an outline of what needed to happen: convert code to C++, write extensible libraries for each component of the robot with its own test program, and finally write the "Robot Brain" to integrate it all. We adopted the practice of using SVN version control using Microsoft Visual Studio 2008 and OpenCV 2.2b4766.

In the last phase we spent time converting all code to C++ console programs with access to the "Common Language Runtime" library provided by Microsoft. Thanks go to Alan for his NXT/Bluetooth packet sending code. In order to be extendible to future projects Ben described the functions used to communicate between the modules in a document called "Application Programmer Interfaces.txt". Ben became involved briefly with the power system layout debugging, then with the group's help eventually got a version of the "Robot Brain" to work. During this time Gehad was heavily involved in debugging and tuning the "Robot Brain" as well as proving that OpenCV routines could be sped up dramatically with techniques described later in this document. Finally, we tested a primitive tempo detection algorithm when someone with the green cap treated it like a drumstick, which was able to work for tempos around 60-70 beats per minute.

To make the project more interesting, we decided to bring Einstein to live by animate him. After several discussions, we came to the agreement of having him animate several human behaviors – walking toward, turning left, right, revulsion, and relax.

Jessie used Pololu Maestro Controller to control the body servos of Einstein. The ready-to-use software was helpful in accelerating the animation process. The combination of creativity imagination and a robust software program were successfully resulted Einstein to have human's behaviors. Walking forward was very direct. Left leg up, right leg down, right arm up and left arm down are the motion that needed to program Einstein. After the moving motions are set, Jessie began to play with the servos and this is where creativity and patient took place. Jessie ensured that the movements are smooth and the servos should be at the right degree so that each movement looks real.

Einstein's body was built by wood, thus he has no flexibility when turning left and right as humans do. To overcome this problem we decided to animate him turning left and right by allocated both of his arms to the direction of turning then returns to his walking forward motion. Revulsion was the hardest part when animating Einstein. We had to design the animation is such a way, so as he is backing up, he would look scared. Revulsion did not only involve Einstein's legs and arm movements, but also the gesture of his face as well.

After testing each behavior of Einstein successfully by using Pololu Maestro Controller, Jessie merged the servo degrees for each movement of each behavior with our main program. We came across some minor bugs when we finally compiled the main program. The degree of movements were off and as a result, Einstein unsuccessfully moved smoothly causing the behaviors to look surreal. Jessie spent hours and hours debugging and finally found a way for Einstein's behavior to move smoothly.

To have a better animation of Einstein, we decided to have him speak. Jessie created a VB program that successfully allows user to insert any text, and Einstein will speak out loud the input content. Since our main program is written in VS C++, we need to manipulate the text to speech program so that it compiles with the main program. We were running out of time and decided to leave it as an opportunity for the next group.

Application Programmer Interfaces

Robot Brain Application Programmer Interface Documentation

by Ben Schaeffer

Thursday March 10, 2011

Overview

This document describes the functions that makeup the Einstein Robot Brain. They are in three components, OpenCV for video, serial for Bluetooth/NXT, and Pololu distribution functions for the Mini Maestro servo controller. Respectively this corresponds to version "OpenCV2.2b4766"(pre-built) files, .Net functions accessed from C++, and "USBWrapper.dll" and "Usc.dll" libraries which need to be listed

in Visual Studio 2008 in the "Common Properties", "References" list. The firmware running on the NXT will need to have the RobotC program called "einsteinNXTAuto.c" running to respond to serial output.

Each component has its own test program and "#define" constant for turning off and on debug output. Our group's hope is that these functions will be portable and adaptable to similar hardware arrangements.

Einstein Video library

check Appendix for implementation (Einstein Video program)

Functions:

VideoTrackingStart

VideoTrackingUpdate

VideoTrackingEnd

This library handles video tracking of objects or humans. At this point only recognizing one person at a time is of interest, and we are limiting object tracking to only bright green colors (a highlighter).

To use the library the calling application first needs to initialize the video library requesting a tracking mode (object or human). If the video camera and preparatory functions all succeed, a point corresponding to screen size in pixels is returned, otherwise the screen size will be 0.

Upon successful initialization the library needs periodic updates, ideally at 30 frames per second or faster. When the frame has something in it that can be detected, a center coordinate is returned along with area in pixels; on failure the area will be 0.

Before exiting the resources allocated to processing video tracking can be freed up by calling this library's function to end tracking.

Einstein Servo Library

check Appendix for implementation (Servo Controller Program)

Functions:

ServoInitialize

ServoUpdate

This library handles animating servos controlled by a Pololu Mini Maestro 18 servo controller (which needs to be connected to the PC during runtime via a USB cable). It is organized around predefined

behaviors which correspond to a sequence of servo positions. These positions can be run once through, holding the last position, or run as an loop continuously until the next behavior is requested. "EinsteinServos.cpp" shows a simple code example of how to use the library.

To use the library the calling application needs to initialize, request a behavior, and then update the servos at (roughly) regular intervals. This interval period will depend on the speed of the PC running the program and might range from 1 to 30 times a second. In order to compensate for this variation, there is a "SetServoBehavior" parameter called "steps" that allows control of the number of steps between each position. For example, if a system was getting updating the servos at 30 frames per second and positions in a behavior needed to be 1 second apart, call "SetServoBehavior" with "steps" set to 30. Use caution with setting "steps" to low values (3 and under) as this potentially could cause excessive power consumption and servo wear and tear.

To customize or use "EinsteinServoLIB" use these guidelines:

1. Store sets of servo positions in "servopositionarray". This array has 42 sets which correspond to one set at index 0 for a random position, one set at index 1 for the neutral position which is used during initialization, and 40 sets for holding 10 behaviors each of which can loop through 4 positions. Expand this as needed, for more total number of behaviors or positions per behavior.

Note #1: disconnected servos or connected servos that are to be left alone need to have corresponding values of -1 in "servopositionarray", index 1, so the library will know not to attempt to connect to them.

Note #2: the random position "servopositionarray", index 0, takes the neutral position at "servopositionarray", index 1, and modifies it plus or minus 100. Only use this feature when all servos have enough space, otherwise the servos may collide with something.

Note #3: at any point in time a new behavior can be requested, and this causes the servo to move from its current position to the first position for the new behavior; some forethought must be given to all sets of servo positions so servos do not collide.

2. Specify sequences of servo position sets in "behaviorarray". This array can be resized as needed as long as each behavior ends with a trailing -1 or -2.

Note #4: repeating a servo position set allows creating a short pause in the middle of a behavior.

3. Name a corresponding constant for each behavior and update the "CountofBehaviors" accordingly.

Bluetooth Library

check Appendix for implementation (Bluetooth)

Functions:

connectSerial

processChar

This library handles sending two-byte command packets to the NXT via Bluetooth. The commands are as follows:

f-forward

s-stop (ramps motors to a stop in 0 - 0.83 seconds depending on speed)

b-backwards

r-right

l-left

e-emergency stop

m-"mad" mode

Except for "m", the "mad" mode which will controls lights and motors on the neck and head, these commands control the four wheels on Einstein's base. Except for the "e" and "s" stops, the motor control commands use the second parameter "magChar" to pass in a power percentage value between 0 and 100. These commands need to be sent regularly, otherwise the NXT will slow to a stop after 1-2 seconds of inactivity. This is a safety precaution that can be modified in the NXT code.

To use the library first call "connectSerial" with the correct com port, typically "com5" or "com6". If successful then continue by calling "processChar" with the appropriate command and magnitude.

The Bluetooth mechanism is not 100% reliable so commands may need to be sent more than once.

Android application (remote control)

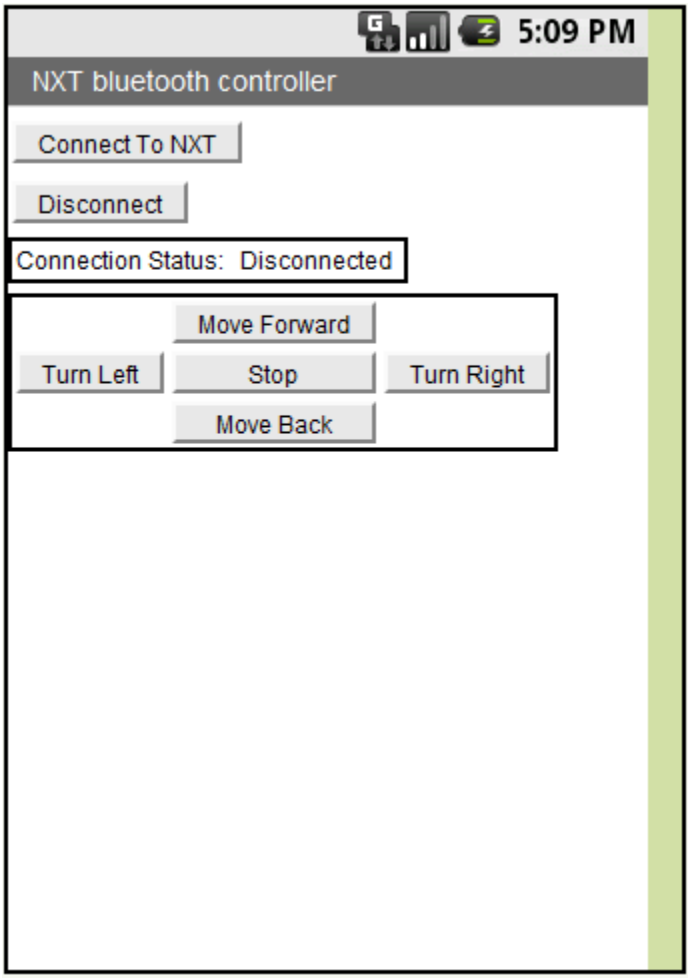


Figure 44: Android Application GUI v.1

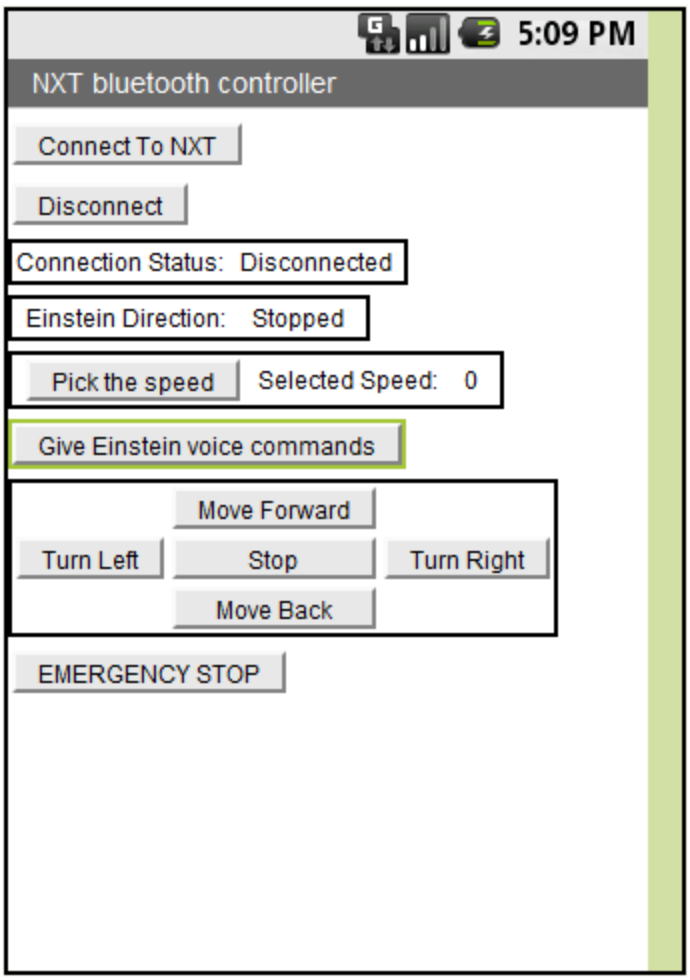


Figure 45: Android Application GUI v.2

CUDA trial

Gehad wanted to give students for next term a base program that will possibly work with CUDA architecture, which will speed up the image processing. In the CUDA sample we have a working code that does template matching with OpenCV once with CUDA and once without CUDA and compares the speed. Our experiment included random image.

To be able to run our experiment you need to have the following:

- OpenCV 2.2 built with CUDA
- CUDA toolkit
- NVIDIA Performance Primitives (NPP) library
- GPU Computing SDK
- Environmental variables to be set

Figure 46: image used in cuda trial (1024 x 768)

Figure 47: template used in cuda trial (61x56)

As we can see from the result below, the GPU processing runs 2.5x faster than cpu version of the same process.

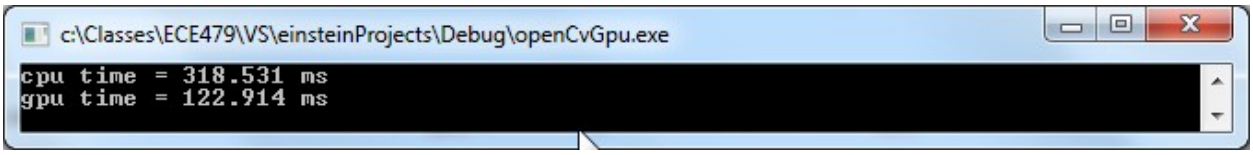


Figure 48: cuda test for random template matching process

We can see how the GPU runs around 2 times faster than the CPU version of template matching. not that this experiment didn't include finding the peaks.

TBB (threading building blocks) trial

The problem with many robotic codes is that most people have a bunch of code that runs sequentially, even though large sections of the code can be run in parallel. Gehad decided to experiment with TBB, which was an open source library from Intel that allowed us to easily parallelize sections of code. In our

experiment we ran the CPU version of template to match the same time that we were running the GPU version of the template matching. Again, this is just to serve as a base program for future students.

To be able to run our experiment you need to have the following:

- OpenCV 2.2 built with CUDA
- TBB library installed
- link the library correctly to your program

We used the task_group class, which allows us to group tasks that can be run in parallel. To make it easier you can run things like the following:

```
task_group g;
g.run([&]{
	// task 1.0
	// task 1.1
}); // spawn a sequential task
g.run([&]{
	// task 2.0
	// task 2.1
}); // spawn another sequential task that can be run in parallel
to taks 1
	g.wait(); // wait for both tasks to complete
```

However, you need to enable lamda expression please follow this tutorial to enable lamda expression.

http://software.intel.com/sites/products/documentation/studio/advisor/en-us/2011/ug_docs/common/adding_the_CPP0X_lambda_function_with_TBB.htm

Our experiment didn't include the lambda expression, although we used the overloaded operator () to call up a run function, which required us to include the function in the class. Please refer to appendix for full code and note that CPUmatch is not a function, it's a class!

```
task_group group;
try
{
	group.run(CPUmatch());
	group.run(GPUmatch());
	group.wait();
} catch (...) {
	;
}
```

Appendix

References

"DC-DC Converters: A Primer," Jacar Electronics Reference Sheet: DCDCCONV.pdf, Jacar Electronics, 2001

Gasperi, Michael. Philo's Home Page. 24 Jan. 2011. Web. 17 Mar. 2011. <http://www.philohome.com/index.htm>.

Huster, Dean. DIY Audio. Posted 16 May 2003. http://www.electro-tech-online.com/general-electronics-chat/1442-running-voltage-regulators-parallel.html

"National Semiconductor | High-performance Analog." 2011. Web. 16 Mar. 2011. <http://www.national.com/analog>.

"Power Supply Regulation." FPGA CPLD and ASIC from Altera. Web. 16 Mar. 2011. <http://www.altera.com/support/devices/power/regulators/pow-regulators.html>.

Code & Pseudocode

Robot Brain Pseudocode

```
//Robot Brain Pseudocode
//by Ben Schaeffer
//Thursday February 27, 2011

Declare Variables //for now treat as global

Declare HumanTracking function
Declare ObjectTracking function
Declare Servo functions
Declare Motor functions

Main Program
      Initialize Everything //memory allocation, camera connection, GUI
      Mode = Nothing
      Time = GetTime(Now)
      Until ExitCondition is True
            if GUI control not equal Mode //GUI is either human or object
tracking
                  if GUI control = human
                        Mode = human
                  else
                        Mode = object
                  ModeReset = True
            Take picture from camera
            NewTime = GetTime(Now)
            ServoUpdate()
            dt = NewTime - Time
            Time = NewTime
            if Mode is human
                  HumanTracking()
            else
                  ObjectTracking()
```

```
      End Until Loop
      Deallocate Everything


------------------------------------------------------------------------
--
//Servo Functions Pseudocode
//I propose that two functions handle servo positions, ServoUpdate and
ServoSetBehavior
//Behaviors (in behavior array):
//    Walking (concurrent with forward motion)
//    Turning Right
//    Turning Left
//    Revulsion (concurrent with backward motion, hands projecting face)
//    Relaxed/At ease (stopped, thinking about something), start with index 0
//    Test (1)

//Each behavior is made up of a sequence of integers which represent indices
in
//an array of position structures that store all servo positions. Each number
is
//very similar to a frame of animation, and this can be running in a loop or
//running once through and then holding some final position. Use a -1 to
indicate
//loop back to beginning, -2 for holding the position. Although this will be
in
//the source code, it ultimately could be migrated to a csv file which can be
//edited in Excel or OpenOffice.

//Also needed in advance is an array of position structures or objects where
each
//structure has 18 servo positions. I recommend using the first position as a
//place holder for a random position at index 0, and the other position
structures
//start with a neutral or initial position which is index 1. if a test mode
//is going to test something other than the servos, then the servos can
simply be
//put in this neutral position (1).

ServoSetBehavior(index, steps) //index of behavior array, steps is number of
            //fractional jumps to take to get to the target position
      if behavior is equal index
            return
      behavior = index // value for walking, turning right, turning left, etc.
      positionindex = 0 // go to the beginning of the behavior array
      servomaximumstep = steps // larger numbers correspond to slower motion
      servocurrentstep = 0
      initialposition18 = currentposition18
      if behaviorarray[behavior][0] equals 0
            put random values for servos at positionarray[0]
      targetposition18 = positionarray [behaviorarray[behavior][0]]

ServoUpdate()
      if servocurrentstep equal servomaximumstep
            if behaviorarray[behavior][positionindex+1] equals -1 //we are in
a looping sequence
                  positionindex = 0
```

```
            else if behaviorarray[behavior][positionindex+1] equals -2 //if
we are holding the last position
                  Return from function
            else
                  Increment positionindex //Increment to next valid index
            if behaviorarray[behavior][positionindex] equals 0 // old random
servo position
                  put new random values for servos at positionarray[0]
            targetposition18 = positionarray [behaviorarray[behavior]
[positionindex]]
            initialposition18 = currentposition18
            servocurrentstep = 0
      increment servocurrentstep
      currentposition18 = weighted sum of initialposition18, targetposition18
                  {based on fraction 'servocurrentstep/servomaximumstep'}
                  // NOTE we could use a cosine or S curve here
      MiniMaestro.SetPosition(currentposition18)

------------------------------------------------------------------------------
--
//RobotC NXT OS Proposed Change Pseudocode
// 1. Loop around 1/30th of a second
// 2. Bluetooth commands are two bytes long
// 3. Each iteration actual speed approaches target speed by some small
constant
// 4. Watchdog timer which sets target speedvariables to 0 if no new Bluetooth
//    commands are encountered within 1-2 seconds. This is an optional safety
//    precaution.
OS()
      Initialization
      ...
      watchdog = 0 //ticks
      const watchdogthreshold = 30 //ticks
      time = now()

      LOOP:
            While time + (1000ms/30) < now() // code must be smart compare <
0 because of 32k limitation
                  Do nothing
            time = time + (1000ms/30)
            if Bluetooth command pending
                  cmd[0,1] = Bluetooth command
                  SetNewTargetSpeeds(cmd)
                  Watchdog = 0
            else
                  if Watchdog > WatchdogThreshold
                        cmd[0] = 's'
                        SetNewTargetSpeeds(cmd)
                  else
                        Watchdog = Watchdog + 1

            if SensorCheck (cmd[0]) //sensorcheck() is going to check if the
command should be aborted
                  //apply to 'f', 'r', 'l'
                  //Emergency stop
                  Target[RightFront] = 0
                  Target[LeftFront] = 0
```

```
                  Target[RightBack] = 0
                  Target[LeftBack] = 0
                  Motor[RightFront] = 0
                  Motor[LeftFront] = 0
                  Motor[RightBack] = 0
                  Motor[LeftBack] = 0
                  //Play sound after robot is stopped
                  PlaySound()//BRIEFLY!

            //UpdateSpeed
            if Motor[RightFront] != Target [RightFront]
                  if Motor[RightFront] < Target [RightFront]
                        Motor[RightFront] = Motor[RightFront] + MotorIncrement
                        //Check if motor has surpassed target value...
                        if Motor[RightFront] > Target [RightFront]
                              Motor[RightFront] = Target [RightFront]
                  else
                        Motor[RightFront] = Motor[RightFront] - MotorIncrement
                        if Motor[RightFront] < Target [RightFront]
                              Motor[RightFront] = Target [RightFront]

            //...Repeat for each motor

      GoTo LOOP //Loop infinitely, acceptable and actually needed for OS
programs


SetNewTargetSpeeds(cmd[])
      if cmd[0] = 'e' //Emergency stop
            Motor[RightFront] = 0
            Motor[LeftFront] = 0
            Motor[RightBack] = 0
            Motor[LeftBack] = 0
            Target [RightFront] = 0
            Target [LeftFront] = 0
            Target [RightBack] = 0
            Target [LeftBack] = 0
      else if cmd[0] = 's' //Normal stop
            Target [RightFront] = 0
            Target [LeftFront] = 0
            Target [RightBack] = 0
            Target [LeftBack] = 0
      else if cmd[0] = 'f'
            Target [RightFront] = cmd[1]
            Target [LeftFront] = cmd[1]
            Target [RightBack] = cmd[1]
            Target [LeftBack] = cmd[1]
      else if cmd[0] = 'b'
            Target [RightFront] = -cmd[1]
            Target [LeftFront] = -cmd[1]
            Target [RightBack] = -cmd[1]
            Target [LeftBack] = -cmd[1]
      else if cmd[0] = 'r'
            Target [RightFront] = -cmd[1]
            Target [LeftFront] = cmd[1]
            Target [RightBack] = -cmd[1]
            Target [LeftBack] = cmd[1]
```

```
      else if cmd[0] = 'l'
            Target [RightFront] = cmd[1]
            Target [LeftFront] = -cmd[1]
            Target [RightBack] = cmd[1]
            Target [LeftBack] = -cmd[1]
      else if cmd[0] = 'm' //MAD Einstein
            MAD(cmd[1])
```

Einstein Brain

```
// einsteinBrain.cpp : main project file.

#include "stdafx.h"

#include "../einsteinBluetooth/bluetoothLIB.h"
//#define __SERVOTESTMODE__
#include "../einsteinServos/einsteinServoLIB.h"
//Defining this variable enables the video tracking output window
#define __VIDEOTESTMODE__

#include "../einsteinVideo/einsteinVideoLIB.h"
#include <ctime>
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>

#define AREATOOLARGEHUMAN (3000)
#define AREATOOSMALLHUMAN (2600)
#define AREATOOLARGEOBJECT (500)
#define AREATOOSMALLOBJECT (30)
#define MAXIMUMVELOCITYNXT (50)
//#define OPENCV_ROOT  "E:\\Program
Files\\OpenCV2.2\\data\\haarcascades\\haarcascade_frontalface_alt_tree
.xml"

void displayDetections(IplImage * pInpImg, CvSeq * pFaceRectSeq);

// We make this value global so everyone can see it.
//
int area0 = 0, area0maximum = 12;
int begin_switch_value = 0;
int bluetooth_switch_value = 1;
int bluetooth_com_port_switch_value = 5;
int mini_maestro_switch_value = 0;
int object_human_switch_value = 1;
int camera_switch_value = 1;
static int drumframe = 0;
using namespace System;

void myTTS(System::String^ s)
{
```

```
// talk or use debug output to console
      Console::WriteLine(s);
}
// pass in height in pixels, time in milliseconds
void addDrumData (int y, float t_ms)
{
      static int height[600];
      static float times_ms[600];// relative time of frame taken in
milliseconds
      static const float four = 4000.0; // 4 seconds
      System::String ^ s;
      //add data here, then examine to see if tempo calculation
warranted
      height [drumframe] = y;
      times_ms[drumframe] = t_ms;
      drumframe++;
      Console::Write("{0} ",t_ms);
      //if there are enough frames to examined and 4 seconds have
passed
      //   and if maximum height v. minimum height > 100
      //         then output tempo message
      if (drumframe > 6 && times_ms[drumframe-1] >= four)
      {
            int i, max, min;
            max = height[0];
            min = max;
            // determine maximum and minimum height
            for(i = 1; i < drumframe; i++)
            {
                  if (max < height[i])
                        max = height[i];
                  else if (min > height[i])
                        min = height[i];
            }
            if (max - min <= 100)
            {
                  s = "Drum detected, not active";
                  myTTS(s);
            }
            else
            {     //determine if steady beats occurred based on crossing
the average
                  float middle = (float)(max+min)*0.5;
                  //Here we define a beat as a crossing of the middle
value with positive slope
                  int i0, i1, i2; //indices for beat beginning, beat
ending will be i + 1
                  // find 2 periods and if close to each other then beat
detected
                  float period0, period1 = -1; //second period initially
invalid time duration
```

```
                    //find a low height, then find crossover of "middle"
                    for(i = 0; i < drumframe && height[i] > middle; i++)
                          ;
                    for(; i < drumframe && height[i] <= middle; i++)
                          ;
                    i0 = i - 1;//found first crossover
                    for(; i < drumframe && height[i] > middle; i++)
                          ;
                    for(; i < drumframe && height[i] <= middle; i++)
                          ;
                    i1 = i - 1;//found second crossover
                    for(; i < drumframe && height[i] > middle; i++)
                          ;
                    for(; i < drumframe && height[i] <= middle; i++)
                          ;
                    i2 = i - 1;//found third crossover
                    // assume failure, check if success actually happened
                    s = "Drumbeat not steady enough";
                    if (i < drumframe)
                    {
                          // find average between crossover times, period
is their difference
                          period0 = 0.5*(times_ms[i1] + times_ms[i1 + 1] -
                                      times_ms[i0] - times_ms[i0 + 1]);
                          period1 = 0.5*(times_ms[i2] + times_ms[i2 + 1] -
                                      times_ms[i1] - times_ms[i1 + 1]);

                          float drumerror;
                          // if periods are 80% similar, then error will be
under 20%,
                          //       then treat as part of steady beat
                          if (period0 > period1)
                                drumerror = (period0 - period1)/period0;
                          else
                                drumerror = (period1 - period0)/period1;

                          if (drumerror < 0.2)
                          {
                                //estimate beats per minute based on
average period
                                int bpm = (int)(60*(0.5*period0 +
0.5*period1)/1000.0);
                                System::String^ b =
System::Int32(bpm).ToString();
                                s = b + " beats per minute detected";
                          }
                    }
                    myTTS(s);
              }
              //finish by preparing for next set of data
              drumframe = 0;
```

```
	}

}

void MyServoSetBehavior(int behavior, int steps)
{
	static int internalbehavior = TestBehavior;
	if (internalbehavior == behavior || internalbehavior ==
TestBehavior)
	{
		ServoSetBehavior(behavior, steps);
		internalbehavior = behavior;
	}
	else
	{
		ServoSetBehavior(TestBehavior, 5);
		internalbehavior = TestBehavior;
	}
}

int main(array<System::String ^> ^args)
{
	//array<System::DateTime^>^ times = gcnew<System::DateTime
^>(600);
	DateTime drumstart;
	Pololu::Usc::Usc^ device;
	SerialPort^ serialPort;

    Console::WriteLine(L"Initializing Robot Brain...");
	// Name the main window
	//
	cvNamedWindow( "Robot Brain Configuration", 0 );
	// Create the trackbar. We give it a name,
	// and tell it the name of the parent window.
	//

	cvCreateTrackbar(
	"Stop | Run",
	"Robot Brain Configuration",
	&begin_switch_value,
	1,
	NULL
	);

	cvCreateTrackbar(
	"Use NXT",
	"Robot Brain Configuration",
	&bluetooth_switch_value,
	1,
	NULL
	);
	cvCreateTrackbar(
```

```
	"NXT COM#",
	"Robot Brain Configuration",
	&bluetooth_com_port_switch_value,
	99,
	NULL
	);
	cvCreateTrackbar(
	"Use Servos",
	"Robot Brain Configuration",
	&mini_maestro_switch_value,
	1,
	NULL
	);
	cvCreateTrackbar(
	"Obj|Human",
	"Robot Brain Configuration",
	&object_human_switch_value,
	1,
	NULL
	);
	cvCreateTrackbar(
	"Camera#",
	"Robot Brain Configuration",
	&camera_switch_value,
	10,
	NULL
	);
//There appears to be some bug in the 'height' parameter
	cvResizeWindow("Robot Brain Configuration", 640, 400);

	while( 1 ) {
		if (cvWaitKey(15)==27) return 0;
		if (begin_switch_value) break;
	}
	cvDestroyWindow( "Robot Brain Configuration");

	//NXT connection through Bluetooth initialization
	if (bluetooth_switch_value)
	{
		System::String^ portName_1 = "com";
		System::String^ portName = portName_1->Concat(portName_1,
bluetooth_com_port_switch_value);

		if(connectSerial(serialPort, portName) == false)
			return -103;
		if (!serialPort)
			return -104;
	}

//If the escape key was not hit then initialize and run program
	//Video initialization
	int area = 0, inputchar = 0;
```

```
	CvPoint center;

	CvPoint screen = CvPoint(VideoTrackingStart(
			(videotrackingtype)object_human_switch_value, 0));
	if (screen.x == 0)
	{
		Console::WriteLine(L"error in starting up video camera!");
		return -101;
	}

	//Servo initialization
	if (mini_maestro_switch_value)
	{
		ServoInitialize(20,20);
		MyServoSetBehavior(TestBehavior, 1);
		// connect to servo controller device

		try
		{
			device = connectToDevice();
		}
		catch (System::Exception^ exception)  // Handle exceptions
by displaying them to the user.
		{
			displayException(exception);
			return -102;
		}
	}

	int servosteps = 15, RelaxedBehaviorServosSteps = 9,
		RevulsionBehaviorServoSteps = 5, WalkingBehaviorServoSteps
= 9;
	int leftthreshold = (7*screen.x)/16;
	int rightthreshold = (9*screen.x)/16;
	int xmagnitude;
	do
	{
		//Check for object
		VideoTrackingUpdate(area, center);

		if (area > 0)
		{
			area0 = 0;//reset counter because area is positive

			if (object_human_switch_value == 1) //human tracking
			{
				// check if human is too close, too far and
centered, else distance is good
				if (area > AREATOOLARGEHUMAN)
				{
					//Retreat backwards
```

```
                              double tmp = log10((double)area) -
log10((double)AREATOOLARGEHUMAN);
                              tmp *= 100.0;
                              if (tmp > MAXIMUMVELOCITYNXT)
                                    tmp = MAXIMUMVELOCITYNXT;

                              behavior = RevulsionBehavior;
                              if (mini_maestro_switch_value)
                              {
                                    servosteps =
RevulsionBehaviorServoSteps;
                                    MyServoSetBehavior(behavior,
servosteps);
                              }
                              if (bluetooth_switch_value)
                              {
                                    //backwards
                                    processChar(serialPort, 'b',
(char)tmp);
                              }
                        }
                        else if (area < AREATOOSMALLHUMAN &&
                              (center.x >= leftthreshold && center.x <=
rightthreshold))
                        {
                              //Walking towards human
                              double tmp =
log10((double)AREATOOSMALLHUMAN) - log10((double)area);
                              tmp *= 100.0;
                              if (tmp > MAXIMUMVELOCITYNXT)
                                    tmp = MAXIMUMVELOCITYNXT;

                              behavior = WalkingBehavior;
                              if (mini_maestro_switch_value)
                              {
                                    servosteps =
WalkingBehaviorServoSteps;
                                    MyServoSetBehavior(behavior,
servosteps);
                              }
                              if (bluetooth_switch_value)
                              {
                                    //forward
                                    processChar(serialPort, 'f',
(char)tmp);
                              }
                        }
                        else// human distance is good
                        {//now relax if in middle, otherwise turn
proportional to human
```

```
                        if (center.x >= leftthreshold && center.x
<= rightthreshold)
                        {
                              //Object detected near center.
                              if (behavior != RelaxedBehavior)
                              {
                                    behavior = RelaxedBehavior;
                                    if (mini_maestro_switch_value)
                                    {
                                          servosteps =
RelaxedBehaviorServosSteps;

      MyServoSetBehavior(behavior, servosteps);
                                    }
                                    if (bluetooth_switch_value)
                                    {
                                          //stop
                                          processChar(serialPort,
's', 0);
                                    }
                              }
                        }
                        //Testing turning right
                        else if (center.x > rightthreshold)
                        {
                              behavior = TurningRightBehavior;
                              xmagnitude = (100 * (center.x -
rightthreshold))/(screen.x-rightthreshold);
                              if (mini_maestro_switch_value)
                              {
                                    servosteps = (10 -
xmagnitude/10) + 3;
                                    MyServoSetBehavior(behavior,
servosteps);
                              }
                              if (bluetooth_switch_value)
                              {
                                    //right
                                    processChar(serialPort, 'r',
xmagnitude);
                              }
                        }
                        else
                        {
                              behavior = TurningLeftBehavior;
                              xmagnitude = (100 * (leftthreshold -
center.x))/(leftthreshold);
                              if (mini_maestro_switch_value)
                              {
                                    servosteps = (xmagnitude/10) +
3;
```

```
                                    MyServoSetBehavior(behavior,
servosteps);
                              }
                              if (bluetooth_switch_value)
                              {
                                    //left
                                    processChar(serialPort, 'l',
xmagnitude);
                              }
                        }
                  }
                  else //object tracking
                  {
                        if (drumframe == 0)
                        {
                              drumstart = DateTime::Now;
                              addDrumData (center.y, 0);
                        }
                        else
                        {
                              System::DateTime difference =
DateTime::Now;
                              System::TimeSpan t_s =
difference.Subtract(drumstart);
                              addDrumData (center.y, (float)
(t_s.Milliseconds +
                                    t_s.Seconds * 1000));
                        }
                  }
            }
            else
            {// all tracking modes disengage when video input disappears
after awhile

                  if (area0 == area0maximum)//if area is 0 for enough
times, relax robot
                  {
                        if (behavior != RelaxedBehavior)
                        {
                              behavior = RelaxedBehavior;
                              if (mini_maestro_switch_value)
                              {
                                    servosteps =
RelaxedBehaviorServosSteps;
                                    MyServoSetBehavior(behavior,
servosteps);
                              }
                        }
                        // separately stop NXT in case prior packet was
lost
```

```
                        if (bluetooth_switch_value)
                        {
                              //stop
                              processChar(serialPort, 's', 0);
                        }
                  }
                  else
                  {
                        if (object_human_switch_value == 0 && area0 == 1)
                              //if more than 1 empty frame cancel tempo
calculation
                              drumframe = 0;
                        area0++; //Counter for finally giving up on
tracked object.
                  }
            }

            if (mini_maestro_switch_value)
            {
                  //Move servos along towards next position
                  ServoUpdate(device);
            }

            //Status display, possibly faster to comment out this line:
#ifdef __VIDEOTESTMODE__

            //printf("area: %d, center: %d %d\n", area, center.x,
center.y );
#endif
            if(Console::KeyAvailable == true)
            {
                  inputchar = Console::ReadKey().KeyChar;

                  if(inputchar != (int)ConsoleKey::Escape)
                  {
                        //Test code, currently disabled
                        /*
                        if (inputchar == (int)('0'))
                        {
                              ;
                        }
                        */
                  }
            }
      }
      while(inputchar != (int)ConsoleKey::Escape );

      VideoTrackingEnd();

      return 0;
}
```

einsteinBluetooth.cpp

```
#include "stdafx.h"
#define __BLUETOOTHTESTMODE__
#include "bluetoothLIB.h"
//namespaces
using namespace System;
using namespace System::IO::Ports;
int main(array<System::String ^> ^args)
{
	SerialPort^ serialPortBT;
	if(tryToConnectSerial(serialPortBT) == false)
		return -1;

	int inputChar;
	while(1)
	{
		if(!serialPortBT)
			return 0;
		if(keyCheckAndProcess(serialPortBT) == false)
			return -2;
	}
	return 0;
}
```

bluetoothLIB.h

```
/*************************************************************************
* Bluetooth High level functions
* Written by: Gehad Shaat
*************************************************************************/
#include "stdafx.h"
using namespace System;
using namespace System::IO::Ports;

/*************************************************************************
* Connect to to serial port (includes bluetooth)
* description:
*     connects to a com port passed
* arguments:
*     uninitialized SerialPort instance (SerialPort^)
*     portname (String^)
* returns:
*     true when successful
*     false when unsuccessful
*************************************************************************/
bool connectSerial(SerialPort^ &serialPort, String^ portName)
```

```
{
	serialPort = gcnew
SerialPort(portName,9600,System::IO::Ports::Parity::None,
8,System::IO::Ports::StopBits::One);
	serialPort->ReadTimeout = 300;
	serialPort->WriteTimeout = 300;
	try
	{
		serialPort->Open();
	}
	catch(...)
	{
		//same as printf or cout
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("error couldn't connect to serial port");
#endif
		delete serialPort;
		return false;
	}
	return true;
}

/*****************************************************************************
* process a input char
* description:
*	connects to a com port passed
* arguments:
*	initialized and connected SerialPort instance (SerialPort^)
*	command character
*	magnitude character
* returns:
*	NA
*****************************************************************************/
void processChar(SerialPort^ serialPort, char cmdChar, char magChar)
{
	try{
	if(cmdChar == 'e')
	{
		array<Byte>^ em = gcnew array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03,
cmdChar, magChar,0x00};
		serialPort->Write(em,0,em->Length);
		delete em;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("emergency stop");
#endif
	}
	else if(cmdChar == 'f')
	{
		array<Byte>^ forward = gcnew
array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03, cmdChar, magChar,0x00};
		serialPort->Write(forward,0,forward->Length);
		delete forward;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("moving forward");
#endif
	}
	else if(cmdChar == 'b')
```

```
	{
		array<Byte>^ back = gcnew
array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03, cmdChar, magChar,0x00};
		serialPort->Write(back,0,back->Length);
		delete back;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("moving back");
#endif
	}
	else if(cmdChar == 'r')
	{
		array<Byte>^ right = gcnew
array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03, cmdChar, magChar,0x00};
		serialPort->Write(right,0,right->Length);
		delete right;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("moving right");
#endif
	}
	else if(cmdChar == 'l')
	{
		array<Byte>^ left = gcnew
array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03, cmdChar, magChar,0x00};
		serialPort->Write(left,0,left->Length);
		delete left;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("moving left");
#endif
	}
	else if(cmdChar == 's')
	{
		array<Byte>^ stop = gcnew
array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03, cmdChar, magChar,0x00};
		serialPort->Write(stop,0,stop->Length);
		delete stop;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("stopping");
#endif
	}
	else if(cmdChar == 'm')
	{
		array<Byte>^ mad = gcnew
array<Byte>{0x07,0x00,0x80,0x09,0x00,0x03, cmdChar, magChar,0x00};
		serialPort->Write(mad,0,mad->Length);
		delete mad;
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("mad!");
#endif
	}
	else if(cmdChar == 27)
	{
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("escape was pressed");
#endif
	}
	else
	{
```

```
#ifdef __BLUETOOTHTESTMODE__
            Console::WriteLine("unrecognized command");
#endif
      }
      }
      catch(TimeoutException^ ex)
      {
            System::DateTime d = System::DateTime::Now;
            d.Add(System::TimeSpan(0,0,0,0,30));
            while(System::DateTime::Compare(d, System::DateTime::Now) > 0)
                  ;
      }
}


/**************************************************************************
* try to connect to a serial coport
* description:
*     ask for a com port untill a connection is established
* arguments:
*     initialized and connected SerialPort instance (SerialPort^)
*     inputchar (int)
* returns:
*     false if the user press q
*     true if a connection was established successfully
**************************************************************************/
bool tryToConnectSerial(SerialPort^ &serialPort)
{
      String^ port;
      // keep asking for port name until a connection is established or q is
pressed
      do
      {
            // reads port name from user e.g com1
#ifdef __BLUETOOTHTESTMODE__
            Console::Write("Please enter the port name for bluetooth or q to
exit: ");
#endif
            port = Console::ReadLine();
            if(port == "q")
                  return false;
      }while(connectSerial(serialPort,port) == false);
      return true;
}

/**************************************************************************
* just check if a key was pressed
* description:
*     process the pressed key
* arguments:
*     initialized and connected SerialPort instance (SerialPort^)
*     inputchar (int)
* returns:
*     false if the connection was not established
*     true otherwise
**************************************************************************/
```

```
bool keyCheckAndProcess(SerialPort^ serialPort)
{
	int inputChar;
	if(!serialPort->IsOpen)
	{
#ifdef __BLUETOOTHTESTMODE__
		Console::WriteLine("serialPort is not open");
#endif
		return false;
	}
	// process the pressed key (doesn't wait for input)
	if(Console::KeyAvailable == true)
	{
		inputChar = Console::ReadKey().KeyChar;
		//Create delay
		DateTime d1 = DateTime::Now;
		d1 = d1.AddMilliseconds(100);
		//Stream character input at 10Hz for one second
		for(int i=0; i<10; i++)
		{
			processChar(serialPort, (char)inputChar, 80);
			while (d1 > DateTime::Now)
				;
			d1 = d1.AddMilliseconds(100);
		}
	}
	return true;
}
```

Servo Controller Program

einsteinServos.cpp

```
#include "stdafx.h"
#include "einsteinServoLIB.h"
using namespace System;
using namespace System::IO::Ports;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Collections::Generic;
//using namespace System::Windows::Forms;
//using namespace System::Data;
//using namespace System::Drawing;
using namespace System::Text;
using namespace Pololu::UsbWrapper;
using namespace Pololu::Usc;


int main(array<System::String ^> ^args)
{
	int inputchar, behavior = RelaxedBehavior, servosteps = 8;
	//If the count of behaviors increases, the next line should be
	// 0 through the CountofBehaviors-1, not "...Press 0-5..."
```

```
      //Console::WriteLine(L"");

      Console::WriteLine(L"Mini Maestro 18 Servo Test\nPress Escape to
exit.\nPress 0-5 for behaviors and f | m | s for step control.");


      ServoInitialize(20,10);
      ServoSetBehavior(TestBehavior, 1);
      // connect to servo controller device
      Usc^ device;
      try
      {
            device = connectToDevice();
      }
      catch (Exception^ exception)  // Handle exceptions by displaying
them to the user.
      {
            displayException(exception);
            return -1;
      }
      do
      {
            //Move servos along towards next position
            ServoUpdate(device);

            //First delay, then check for a keypress
            DateTime d1 = DateTime::Now;
            d1 = d1.AddMilliseconds(100);//1/10 seconds, change as
desired
            while (d1 > DateTime::Now)
                  ;

            if(Console::KeyAvailable == true)
            {
                  inputchar = Console::ReadKey().KeyChar;

                  if(inputchar != (int)ConsoleKey::Escape)
                  {
                        //Test behavior code
                        //(larger number of steps means slower average
speed of motion)
                        if (inputchar >= (int)('0') && inputchar <=
CountofBehaviors - 1 + (int)('0'))
                        {
                              behavior = inputchar - (int)('0');
                              ServoSetBehavior(behavior, servosteps);
                        }
                        //Test different rates of position changing
                        else if (inputchar == 'f')
                        {
                              servosteps = 3;
                              ServoSetBehavior(behavior, servosteps);
```

```
                        }
                        else if (inputchar == 'm')
                        {
                              servosteps = 8;
                              ServoSetBehavior(behavior, servosteps);
                        }
                        else if (inputchar == 's')
                        {
                              servosteps = 13;
                              ServoSetBehavior(behavior, servosteps);
                        }
                  }
            }
      }
      while(inputchar != (int)ConsoleKey::Escape );

      return 0;
}
```

einsteinServoLIB.h

```
#include "stdafx.h"
#include <stdlib.h>

using namespace System;
using namespace System::IO::Ports;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Collections::Generic;
//using namespace System::Windows::Forms;
//using namespace System::Data;
//using namespace System::Drawing;
using namespace System::Text;
using namespace Pololu::UsbWrapper;
using namespace Pololu::Usc;

//Behaviors:
#define TestBehavior (0)
#define     WalkingBehavior (1)
#define     TurningLeftBehavior (2)
#define     TurningRightBehavior (3)
#define     RevulsionBehavior (4)
#define     RelaxedBehavior (5)
#define CountofBehaviors (RelaxedBehavior + 1) // change as needed
//Each behavior is made up of a sequence of positions (see
'servopositionarray'). Each
//number is similar to a frame of animation which can run in a loop or
//once through (and then holding some final position). Use a -1 to indicate
//loop back to beginning, -2 for holding the position. Although this will be
in
//the source code, it ultimately could be migrated to a csv file which can be
//edited in Excel or OpenOffice.
int behaviorarray[CountofBehaviors][32]=
```

```
{{1,-2,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},//TestBehavior
{2,3,4,5,-1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},//WalkingBehavior
{6,7,8,9,-1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},//TurningLeftBehavior
{10,11,12,13,-
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},//TurningRightBehavi
or
{14,15,16,17,-
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},//RevulsionBehavior
{1 ,1 ,0 ,-1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};//RelaxedBehavior

//It was requested that each servo have its own flag for inclusion in the
random
//list, so in keeping with the 'servopositionarray' to enable a servo for
random
//positioning make it non-negative
int servopermittedrandomlist[18] =
{ 0, 0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};

//The following array lists sets of servo positions that represent a puppet
pose.
//These poses are referred to in the above BehaviorArray. The first position
as a
//place holder for a random position (at index 0), and the other position
structures
//start with a neutral or initial position which is index 1. If a test mode
//is going to test something other than the servos, then the servos can
simply be
//put in this neutral position (i.e. servopositionarray[1][0..17]).
int servopositionarray[42][18] =
{{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //random
{1593,1285,992,1217,1745,-1,992,2000,1398,-1,-1,-1,1177,-
1,1324,1843,1540,1432}, //neutral position
//{1593,1285,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1}, //neutral
position for testing 2 servos
{1593,1285,2000,817,1745,-1,992,2000,1398,-1,-1,-1,1553,-
1,1192,1843,1540,1432}, //2 - Forward frame1
{1593,1585,996,1596,1392,-1,2000,2000,1060,-1,-1,-1,1202,-
1,1378,992,1540,1432}, //3 - Forward frame2
{1593,1285,2000,817,1745,-1,992,2000,1398,-1,-1,-1,1553,-
1,1192,1843,1540,1432}, //4 - Forward frame3
{1593,1285,996,1596,1392,-1,2000,2000,1060,-1,-1,-1,1202,-
1,1378,992,1540,1432}, //5 - Forward frame4

{1598,1574,992,817,1745,-1,1080,2000,1398,-1,-1,-1,1212,-
1,1163,1960,1916,1432}, //6 - Left turn frame1
{1593,1585,996,1596,1392,-1,2000,2000,1060,-1,-1,-1,1202,-
1,1378,992,1540,1432}, //7 - Left turn frame2
{1593,1285,2000,817,1745,-1,992,2000,1398,-1,-1,-1,1553,-
1,1192,1843,1540,1432}, //8 - Left turn frame3
{1593,1285,996,1596,1392,-1,2000,2000,1060,-1,-1,-1,1202,-
1,1378,992,1540,1432}, //9 -Left turn frame4
```

```
{1540,1001,992,1657,1486,-1,992,2000,1368,-1,-1,-1,1305,-
1,1334,1843,1094,1432}, //10 -Right turn frame1
{1593,1285,2000,817,1745,-1,992,2000,1398,-1,-1,-1,1553,-
1,1192,1843,1540,1432}, //11 - Right turn frame2
{1593,1585,996,1596,1392,-1,2000,2000,1060,-1,-1,-1,1202,-
1,1378,992,1540,1432}, //12 - Right turn frame3
{1593,1285,2000,817,1745,-1,992,2000,1398,-1,-1,-1,1553,-
1,1192,1843,1540,1432}, //13 - Right turn frame4

{1593,1320,1740,1217,1745,-1,1540,1432,1853,-1,-1,-1,1701,-
1,1153,1114,1334,1584}, //14 - Revulsion frame1
{1510,1324,1701,1471,1456,-1,1275,992,1338,-1,-1,-1,1280,-
1,1339,114,1461,1329}, //15 - Revulsion frame2
{1000,1021,1740,1217,1745,-1,1540,1432,1853,-1,-1,-1,1701,-
1,1153,1114,1334,1584}, //16 - Revulsion frame3
{1045,1525,1740,1217,1745,-1,1540,1432,1353,-1,-1,-1,1701,-
1,1153,1114,1334,1584}, //17 - Revulsion frame4
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //18
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //19
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //20
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //21
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //22
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //23
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //24
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //25
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //26
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //27
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //28
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //29
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //30
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //31
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //32
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //33
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //34
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //35
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //36
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //37
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //38
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //39
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}, //40
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}}; //41

static int behavior=-1,// initially invalid, recommended to initialize through
//calling ServoSetBehavior()
positionindex = 0, // go to the beginning of the behavior array
currentposition[18], // recommended to initialize to servopositionarray[1]
values
initialposition[18],
targetpositionindex = 1; // setup to start with adjustment towards neutral
position
// private variables
static float servostep_f = 0.1f; //floating point representation of servo
step, value between 0 and 1
static float servocurrent_f = 0.0; //current distance to next position, value
between 0 and 1
```

```
/
*****************************************************************************
*********
* Set the behavior routine for the servos
* description:
*     Prepares servo structures for puppet actions (like walking or turning)
* arguments:
*     index: value in behavior array, i.e. WalkingBehavior
*     steps: number of intermediate positions to insert between the predefined
*           in the position array. The more steps requested, the slower the
overall
*           movement will become; recommended values are 3 through 15.
*****************************************************************************
*********/
void ServoSetBehavior(int index=TestBehavior, int steps=13)
{
      servostep_f = 1.0f/steps;
      if (behavior==index)
      {
            return;
      }
      behavior = index;// value for walking, turning right, turning left, etc.
      positionindex = 0; // go to the beginning of the behavior array

      for (int i = 0; i < 18; i++)
      {
            //Start from wherever the current position is...
            initialposition[i] = currentposition[i];
            //Check for random position, and if needed change from old random
values
            if (behaviorarray[behavior][0] == 0)
            {
                  servopositionarray[0][i] = servopositionarray[1][i];
                  if (servopermittedrandomlist[i] >= 0)
                        servopositionarray[0][i] += -100 + (rand()%200);
                  //Safety check to keep values positive
                  if (servopositionarray[0][i] < 0)
                        servopositionarray[0][i] = 0;
            }
      }
      targetpositionindex = behaviorarray[behavior][0];
}

//Begin Pololu functions

/// <summary>
/// Displays an exception to the user by popping up a message box.
/// </summary>
void displayException(Exception^ exception)
{
      StringBuilder^ stringBuilder = gcnew StringBuilder();
      do
      {
            stringBuilder->Append(exception->Message + "  ");
            if (exception->GetType() == Win32Exception::typeid)
            {
```

```
                stringBuilder->Append("Error code 0x" +
((Win32Exception^)exception)->NativeErrorCode.ToString("x") + ".  ");
            }
            exception = exception->InnerException;
      }
      while (exception != nullptr);
      //                MessageBox::Show(stringBuilder->ToString(), this-
>Text, MessageBoxButtons::OK, MessageBoxIcon::Error);
}

/// <summary>
/// Connects to a Maestro using native USB and returns the Usc object
/// representing that connection.  When you are done with the
/// connection, you should delete it using the "delete" statement so
/// that other processes or functions can connect to the device later.
/// </summary>
Usc^ connectToDevice()
{
      // Get a list of all connected devices of this type.
      List<DeviceListItem^>^ connectedDevices = Usc::getConnectedDevices();

      for each (DeviceListItem^ dli in connectedDevices)
      {
            // If you have multiple devices connected and want to select a
particular
            // device by serial number, you could simply add a line like this:
            //   if (dli.serialNumber != "00012345"){ continue; }

            Usc^ device = gcnew Usc(dli); // Connect to the device.
            return device;             // Return the device.
      }
      throw gcnew Exception("Could not find device.  Make sure it is plugged
in to USB " +
            "and check your Device Manager (Windows) or run lsusb (Linux).");
}

/// <summary>
/// Attempts to set the target (width of pulses sent) of a channel.
/// </summary>
/// <param name="channel">Channel number from 0 to 23.</param>
/// <param name="target">
///   Target, in units of quarter microseconds.  For typical servos,
///   6000 is neutral and the acceptable range is 4000-8000.
/// </param>
Void TrySetTarget(Byte channel, UInt16 target, Usc^ device)
{
      try
      {
            device->setTarget(channel, target<<2);
      }
      catch (Exception^ exception)  // Handle exceptions by displaying them
to the user.
      {
            displayException(exception);
      }
      /*
      finally  // Do this no matter what.
```

```
	{
		delete device;  // Close the connection so other processes can
use the device.
	}
	*/
}

void ServoInitialize(UInt16 delSpeed, UInt16 delAcceleration)
{
	extern int servopositionarray[][18];
	Usc^ device;
	int i;
	try
	{
		device = connectToDevice();
		try
		{
			//Start with slow speed and acceleration
			for(i = 0; i < 18; i++)
			{
				if(servopositionarray[1][i] >= 0)
				{
					device->setSpeed(i, 4);
					device->setAcceleration(i, 4);
				}
			}
			//Move to the neutral position
			ServoSetBehavior();
			//Wait till servo's have gotten into the neutral position
(~2.5 seconds)
			DateTime d1 = DateTime::Now;
			d1 = d1.AddMilliseconds(2500);
			while (d1 > DateTime::Now)
				;

			//Change speed and acceleration to requested values
			for(i = 0; i < 18; i++)
			{
				if(servopositionarray[1][i] >= 0)
				{
					device->setSpeed(i, delSpeed);
					device->setAcceleration(i, delAcceleration);
					//Initialize servo position
					initialposition[i] = currentposition[i] =
servopositionarray [1][i];
				}
			}
		}
		catch (Exception^ exception)  // Handle exceptions by displaying
them to the user.
		{
			Console::WriteLine(exception->ToString());
			Console::WriteLine(L"Exception: A servo failed to set
speed.");
		}
	}
```

```
      catch (Exception^ exception)  // Handle exceptions by displaying them
to the user.
      {
            Console::WriteLine(exception->ToString());
            Console::WriteLine(L"Exception: couldn't connect to servo
device");
      }
      finally  // Do this no matter what.
      {
            delete device;  // Close the connection so other processes can
use the device.
      }
}

//End Pololu functions

/
*****************************************************************************
*********
* Update the position of the servos
* description:
*     Moves the servos according to the current behavior; works best when
called between
*       once and 30 times a second.
*****************************************************************************
*********/
void ServoUpdate(Usc^ device)
{
      int i = 0;
      if (servocurrent_f > 0.99)
      {
            if (behaviorarray[behavior][positionindex + 1] == -1) //we are in
a looping sequence
                  positionindex = 0;
            else if (behaviorarray[behavior][positionindex + 1] == -2) //if
we are holding the last position
                  return;
            else
                  ++positionindex; //Increment now that it has been
determined there is a valid position
            if (behaviorarray[behavior][positionindex] == 0) // old random
servo position
                  for (i = 0; i < 18; i++)
                  {     //put new random values for servos at
servopositionarray[0]
                        servopositionarray[0][i] = servopositionarray[1][i];
                        if (servopermittedrandomlist[i] >= 0)
                              servopositionarray[0][i] += -100 + (rand()%200);
                        //Safety check to keep values positive
                        if (servopositionarray[0][i] < 0)
                              servopositionarray[0][i] = 0;
                  }
                  targetpositionindex = behaviorarray[behavior]
[positionindex];
                  for (int i = 0; i < 18; i++)
                  {
                        //Start from wherever the current position is...
```

```
                        initialposition[i] = currentposition[i];
                }
                servocurrent_f = 0.0;
        }
        servocurrent_f += servostep_f;
        for (i = 0; i < 18; i++)
        {
                //disable negative valued servos (checking neutral position)
                if (servopositionarray[1][i] >= 0)
                {
                        //currentposition18 = weighted sum of initialposition18,
targetposition18
                        //based on fraction 'servocurrentstep/servomaximumstep'
                        // NOTE we could use a cosine or S curve here
                        currentposition[i] =
                                (int)
(servocurrent_f*servopositionarray[targetpositionindex][i]
                        + (1.0 - servocurrent_f)*initialposition[i]);
                        //Safety precaution to keep values non-negative
                        if (currentposition[i] < 0)
                                currentposition[i] = 0;
                        TrySetTarget(i, currentposition[i], device);

#ifdef __SERVOTESTMODE__
                        Console::Write(L"{0:D}:", i + 1);
                        Console::Write(L"{0:D} ", currentposition[i]);
#endif

                        //MiniMaestro.SetPosition(currentposition[i], i);
                }
        }
        Console::WriteLine(L",");
}
```

Einstein Video program

einsteinVideo.cpp

```
#include "stdafx.h"
//Defining this variable enables the video tracking output window
#define __VIDEOTESTMODE__
#include "einsteinVideoLIB.h"

int main(array<System::String ^> ^args)
{
        int inputchar, area;
        CvPoint center;
        Console::WriteLine(L"Video Tracking Test\nPress 0-1 for
mode(OBJECTTRACKING, HUMANTRACKING,...)");
        while(Console::KeyAvailable != true)
                ;
        inputchar = Console::ReadKey().KeyChar-'0';
        if (!(inputchar == 0||inputchar == 1))
```

```
	{
		Console::WriteLine(L"Only 0-1 valid!");
		return -100;
	}
	CvPoint screen =
CvPoint(VideoTrackingStart((videotrackingtype)inputchar, 0));
	if (screen.x == 0)
	{
		Console::WriteLine(L"error in starting up video camera!");
		return -101;
	}
	do
	{
		//Move servos along towards next position
		VideoTrackingUpdate(area, center);
		printf("area: %d, center: %d %d\n", area, center.x,
center.y );

		//First delay, then check for a keypress
		//DateTime d1 = DateTime::Now;
		//d1 = d1.AddMilliseconds(100);//1/10 seconds, change as
desired
		//while (d1 > DateTime::Now)
		//	;

		if(Console::KeyAvailable == true)
		{
			inputchar = Console::ReadKey().KeyChar;

			if(inputchar != (int)ConsoleKey::Escape)
			{
				//Test code, currently disabled
				/*
				if (inputchar >= (int)('0') && inputchar <=
CountofBehaviors - 1 + (int)('0'))
				{
					;
				}
				*/
			}
		}
	}
	while(inputchar != (int)ConsoleKey::Escape );
	VideoTrackingEnd();
}
```

einsteinVideoLib.h

```
typedef enum tag_videotrackingtype {OBJECTTRACKING, HUMANTRACKING}
videotrackingtype; //expand as needed
static videotrackingtype mode;
```

```
#include <ctime>
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <OpenCV2/OpenCV.hpp>
using namespace System;
using namespace System::Threading;
using namespace cv;
#include <stdio.h>
#include <stdlib.h>
#define DETECT_FACE "/../data/haarcascades/haarcascade_frontalface_alt.xml"
//#define DETECT_FACE "E:\\Program
Files\\OpenCV2.2b4766\\data\\haarcascades\\haarcascade_frontalface_alt.xml"
/*
http://www.xavigimenez.net/blog/2010/02/face-detection-how-to-find-faces-
with-OpenCV/
*/

static VideoCapture cam;
static CascadeClassifier  faceCascade;
static      vector<Rect> faces;
static      Mat frame;
static CvPoint screendimensions;
static CvCapture* capture = 0;
static IplImage* frame2 = 0;
static IplImage* image = 0;

void detectAndDraw( Mat& img, CascadeClassifier& cascade, double scale, int
&retarea, CvPoint &retcenter)
{
      retarea = 0;
      int i = 0;
      double t = 0;
      vector<Rect> faces;
      const static Scalar colors[] =
{ CV_RGB(0,0,255),CV_RGB(0,128,255),CV_RGB(0,255,255),CV_RGB(0,255,0),
CV_RGB(255,128,0),CV_RGB(255,255,0),CV_RGB(255,0,0),CV_RGB(255,0,255)} ;
      Mat gray, smallImg( cvRound (img.rows/scale), cvRound(img.cols/scale),
CV_8UC1 );
      cvtColor( img, gray, CV_BGR2GRAY );
      resize( gray, smallImg, smallImg.size(), 0, 0, INTER_LINEAR );
      equalizeHist( smallImg, smallImg );
      t = (double)cvGetTickCount();
      cascade.detectMultiScale(smallImg, faces,
            1.1, 2, 0
            |CV_HAAR_FIND_BIGGEST_OBJECT
            |CV_HAAR_DO_ROUGH_SEARCH
            |CV_HAAR_DO_CANNY_PRUNING
            |CV_HAAR_SCALE_IMAGE
            ,
            Size(40, 40) );
      t = (double)cvGetTickCount() - t;
#ifdef __VIDEOTESTMODE__
      printf( "detection time = %g ms\n", t/
((double)cvGetTickFrequency()*1000.) );
#endif
```

```
	for( vector<Rect>::const_iterator r = faces.begin(); r != faces.end();
r++, i++ )
	{
		Mat smallImgROI;
		Point center;
		Scalar color = colors[i%8];
		int radius;
		center.x = cvRound((r->x + r->width*0.5)*scale);
		center.y = cvRound((r->y + r->height*0.5)*scale);
		if (!i)
		{
			retarea = r->width*r->height;
			retcenter.x = center.x;
			//make y coordinate normal
			retcenter.y = screendimensions.y - center.y;
		}
		radius = cvRound((r->width + r->height)*0.25*scale);
		circle( img, center, radius, color, 3, 8, 0 );
	}
#ifdef __VIDEOTESTMODE__
	cv::imshow( "result", img );
	cv::waitKey(1);
#endif
}

CvPoint VideoTrackingStart(videotrackingtype moderequested, int cameranumber){
	mode = moderequested;
	CvPoint ret;
	ret.x=0;
	ret.y=0;
	if (moderequested == HUMANTRACKING)
	{
		//pcam=NULL;

		// connect to camera
		cam = VideoCapture(cameranumber);

		if(cam.isOpened() == false)
		{
			Console::WriteLine("Couldn't connect a camera");
			return ret;
		}
		//pcam = &cam;
		// to globalize the use of path instead of changing it
		char* OpenCV_path = getenv("OPENCV_LIB_DIR");
		const char* classifier = DETECT_FACE;
		std::string classifier_path(OpenCV_path);
		classifier_path.append (classifier);
		if(faceCascade.load(classifier_path.c_str()) == false)
		{
			Console::WriteLine("couldn't load classifier\n");
			return ret;
		}
		while(cam.read(frame) == false);
	//for testing
		//cv::imshow("video", frame);
		//cv::waitKey(1);
```

```
            ret.x = frame.cols;
            ret.y = frame.rows;
            screendimensions = ret;
      }
      else if (moderequested == OBJECTTRACKING)
      {
            // Initialize capturing live feed from the camera
            capture = cvCaptureFromCAM(cameranumber);

            // Couldn't get a device? Throw an error and quit
            if(!capture)
            {
                  Console::WriteLine("Couldn't connect a camera");
                  return ret;
            }
#ifdef __VIDEOTESTMODE__
            // The two windows we'll be using

            cvNamedWindow("video");
            cvNamedWindow("thresh");
#endif

            // First create a place, 'image', which holds the modified frame
            // (Because the cvQueryFrame function says not to modify the
returned IplImage
            // each time a 'frame' is retrieved it will be copied into
'image')
            frame2 = cvQueryFrame(capture);
            // Checking for video capture frame retrieval functionality...
            if (!frame2)
            {
                  Console::WriteLine("Could not retrieve video frame...\n");
                  return ret;
            }
            ret.x = frame2->width;
            ret.y = frame2->height;
            image = cvCreateImage(cvGetSize(frame2), frame2->depth, frame2-
>nChannels);
            cvCopy( frame2, image, 0 );
      }
      return ret;
}

IplImage* GetThresholdedImage(IplImage* img)
{
      // allocate memory for HSV version of image
      IplImage* imgHSV = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 3);
      // convert original image into a HSV version in the allocated space
      cvCvtColor(img, imgHSV, CV_BGR2HSV); //CV_BGR2HSV possible
      // perform thresholding
      IplImage* imgThreshed = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
      cvInRangeS(imgHSV, cvScalar(43, 118, 118), cvScalar(77, 255, 255),
imgThreshed);
      // deallocate memory used for HSV version of image
      cvReleaseImage(&imgHSV);
      // eliminate noise
      cvErode(imgThreshed, imgThreshed, 0, 1);
```

```
	// return the thresholded version of the image
	return imgThreshed;
}

void VideoTrackingUpdate(int &area, CvPoint &center){
	area = 0;
	if (mode == HUMANTRACKING)
	{
		while(cam.read(frame) == false)
			;
		detectAndDraw(frame,faceCascade,3, area, center);
	}
	else if (mode == OBJECTTRACKING)
	{// modified from http://www.aishack.in/2010/07/tracking-colored-objects-in-OpenCV/

		//Now that we have a video frame get object information (yellow)
		IplImage* imgYellowThresh = GetThresholdedImage(frame2);

		//You first allocate memory to the moments structure, and then you calculate the various moments.
		//And then using the moments structure, you calculate the two first order moments (moment10 and moment01) and the zeroth order moment (area).

		// Calculate the moments to estimate the position of the ball
		CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
		cvMoments(imgYellowThresh, moments, 1);

		// The actual moment values
		double moment10 = cvGetSpatialMoment(moments, 1, 0);
		double moment01 = cvGetSpatialMoment(moments, 0, 1);
		double area2 = cvGetCentralMoment(moments, 0, 0);
		int size = (int)(0.5 + 1.5*(pow(area2, 0.5)));
		delete moments;//Ben: Although this works I personally do not like mixing malloc and delete, C VS. C++

		//Dividing moment10 by area gives the X coordinate of the yellow ball, and similarly,
		//dividing moment01 by area gives the Y coordinate.

		//Now, we need some mechanism to be able to store the previous position.
		//We do that using static variables:

		// Holding the last and current ball positions
		static int posX = 0, posX_ = 0;
		static int posY = 0, posY_ = 0;

		int lastX = posX;
		int lastY = posY;

		center.x = posX = moment10/area2;
		center.y = posY = moment01/area2;

		//The current position of the ball is stored in posX and posY, and the previous location
```

```
            //is stored in lastX and lastY.
#ifdef __VIDEOTESTMODE__

            // Print it out for debugging purposes
            //printf("position (%d,%d)\n", posX, posY);

            // We want to draw a line only if its a valid position
            if(lastX>0 && lastY>0 && posX>0 && posY>0)
            {
                  // Draw a cross hair and point to the center
                  cvLine(image, cvPoint(posX, posY + size), cvPoint(posX,
posY - size), cvScalar(0,0,255), 2);
                  cvLine(image, cvPoint(posX + size, posY), cvPoint(posX -
size, posY), cvScalar(0,0,255), 2);
                  cvLine(image, cvPoint(posX, posY), cvPoint(image->width>>1,
image->height>>1), cvScalar(0,0,0), 1);
            }

            //We simply create a line from the previous point to the current
point, of yellow colour and a width of 5 pixels.

            //The if condition prevents any invalid points from being drawn
on the screen.
            //(Just try taking the yellow object out of the screen once the
program is done… you'll see what I mean).

            //Once all of this processing is over, we combine the scribble
and the captured frame:

            cvShowImage("thresh", imgYellowThresh);
            cvShowImage("video",  image);
            cv::waitKey (1);
#endif

            // Release the thresholded image+moments... we need no memory
leaks.. please
            cvReleaseImage(&imgYellowThresh);

            // Keep checking for video capture frame retrieval
functionality...
            frame2 = cvQueryFrame(capture);
            if (!frame2)
            {
                  area = 0;
                  Console::WriteLine("Could not retrieve video frame...\n");
                  return;
            }
            // Prepare for another loop iteration
            cvCopy( frame2, image, 0 );
            area = (int)area2;
      }
}

void VideoTrackingEnd(){
      if (mode == HUMANTRACKING)
      {
//   deallocation goes here
```

```
	}
	else if (mode == OBJECTTRACKING)
	{
		//Free resources in reverse order of allocation
		cvReleaseImage(&image);

		// We're done using the camera. Other applications can now use it
		cvReleaseCapture(&capture);
	}
}
```

NXT Program

```
//Robot Brain Pseudocode implementation
// written by: Gehad Shaat

#pragma config(Hubs, S1, HTMotor, HTMotor, HTMotor, none)
#pragma config(Sensor, S2, LeftSensor, sensorSONAR)
#pragma config(Sensor, S3, RightSensor, sensorSONAR)
#pragma config(Sensor, S4, FrontSensor, sensorSONAR)
#pragma config(Motor, mtr_S1_C1_1, LeftBack,   tmotorNormal, PIDControl,
reversed, encoder)
#pragma config(Motor, mtr_S1_C1_2, RightBack,  tmotorNormal, PIDControl,
encoder)
#pragma config(Motor, mtr_S1_C2_1, RightFront, tmotorNormal, PIDControl,
encoder)
#pragma config(Motor, mtr_S1_C2_2, LeftFront,  tmotorNormal, PIDControl,
reversed, encoder)
#pragma config(Motor, mtr_S1_C3_1, Neck,       tmotorNormal, PIDControl,
encoder)
#pragma config(Motor, mtr_S1_C3_2, motorI,     tmotorNormal, PIDControl,
encoder)

#define SPEED 50
#define DISTANCE 29
#define FORWARD 0
#define BACK 1
#define LEFT 2
#define RIGHT 3
#define delay_before_moving 200
#define delay_after_moving 200

#define MotorIncrement 3
#define watchDogThresh 60
int TargetRightFront;
int TargetLeftFront;
int TargetRightBack;
int TargetLeftBack;

int currentRightFront;
int currentLeftFront;
int currentRightBack;
int currentLeftBack;

char watchDog = 0; // will be used as integer
```

```
// variables for bluetooth
//char cmd0; // commands received from bluetooth
//char cmd1;
const int kMaxSizeOfMessage = 4;
const TMailboxIDs kQueueID = mailbox1;

void mad() //This motor controls his neck rotation and eye color change.
{
      /*motor[Neck]=-01;
      wait1Msec (300);
      motor[Neck]=05;
      wait1Msec (700);
      motor[Neck]=-01;
      wait1Msec (800);
      motor[Neck]=0;
      */
      return;
}
void SetNewTargetSpeeds(char cmd0, char cmd1)
{
  int sped = (int)cmd1;
      if(cmd0 == 'e') //Emergency stop
      {
            motor[RightFront] = 0;
            motor[LeftFront] = 0;
            motor[RightBack] = 0;
            motor[LeftBack] = 0;
            TargetRightFront = 0;
            TargetLeftFront = 0;
            TargetRightBack = 0;
            TargetLeftBack = 0;
            currentRightFront = 0;
            currentLeftFront = 0;
            currentRightBack = 0;
            currentLeftBack = 0;
      }
      else if (cmd0 == 's') //Normal stop
      {
            TargetRightFront = 0;
            TargetLeftFront = 0;
            TargetRightBack = 0;
            TargetLeftBack = 0;
      }
      else if (cmd0 == 'f')
      {
            TargetRightFront = sped;
            TargetLeftFront = sped;
            TargetRightBack = sped;
            TargetLeftBack = sped;
      }
      else if (cmd0 == 'b')
      {
            TargetRightFront = -sped;
            TargetLeftFront = -sped;
            TargetRightBack = -sped;
            TargetLeftBack = -sped;
      }
```

```
      else if (cmd0 == 'r')
      {
            TargetRightFront = -sped;
            TargetLeftFront = sped;
            TargetRightBack = -sped;
            TargetLeftBack = sped;
      }
      else if (cmd0 == 'l')
      {
            TargetRightFront = sped;
            TargetLeftFront = -sped;
            TargetRightBack = sped;
            TargetLeftBack = -sped;
      }
      else if (cmd0 == 'm') //MAD Einstein
      {
            mad();
      }
      if(sped > 0)
      {
        eraseDisplay();
        nxtDisplayCenteredTextLine(3, "cmd1= %d ", cmd1);
      }
}


bool checkBTLinkConnected()
{
  if (nBTCurrentStreamIndex >= 0)
  {
    eraseDisplay();
    return true;
  }
  eraseDisplay();
  nxtDisplayCenteredTextLine(3, "BT not");
  nxtDisplayCenteredTextLine(4, "Connected");
  return false;
}

void bluetoothCheck(char &cmd)
{
  TFileIOResult nBTCmdRdErrorStatus;
  int nSizeOfMessage;
  ubyte nRcvBuffer[kMaxSizeOfMessage];

  nSizeOfMessage = cCmdMessageGetSize(kQueueID);
  if (nSizeOfMessage == 0)
  {
            if(watchDog > watchDogThresh)
            {
                  SetNewTargetSpeeds('s', 0);
            }
            else
            {
                  watchDog++;
            }
        return;           // No more message this time
```

```
  }
  watchDog = 0;
  if (nSizeOfMessage > kMaxSizeOfMessage)
        nSizeOfMessage = kMaxSizeOfMessage;
  nBTCmdRdErrorStatus = cCmdMessageRead(nRcvBuffer, nSizeOfMessage,kQueueID);
  if(nBTCmdRdErrorStatus == 0)
  {
            cmd = nRcvBuffer[0];
            // set global commands
            //cmd0 = nRcvBuffer[0];
            //cmd1 = nRcvBuffer[1];
            // set target speeds
            SetNewTargetSpeeds(nRcvBuffer[0],nRcvBuffer[1]);
  }

}


void updateSpeed()
{
      // for right front motor
      if (currentRightFront != TargetRightFront)
      {
            if (currentRightFront < TargetRightFront)
            {
              currentRightFront = currentRightFront + MotorIncrement;
                  //Check if motor has surpassed target value...
                  if (currentRightFront > TargetRightFront)
                  {
                        currentRightFront = TargetRightFront;
                  }
            }
            else
            {
              currentRightFront = currentRightFront - MotorIncrement;
                  if (currentRightFront < TargetRightFront)
                  {
                        currentRightFront = TargetRightFront;
                  }
            }
      }

      // for left front motor
      if (currentLeftFront != TargetLeftFront)
      {
            if (currentLeftFront < TargetLeftFront)
            {
                  currentLeftFront = currentLeftFront + MotorIncrement;
                  //Check if motor has surpassed target value...
                  if (currentLeftFront > TargetLeftFront)
                  {
                        currentLeftFront = TargetLeftFront;
                  }
            }
            else
            {
                  currentLeftFront = currentLeftFront - MotorIncrement;
```

```
                if (currentLeftFront < TargetLeftFront)
                {
                      currentLeftFront = TargetLeftFront;
                }
          }
    }

    // for right back motor
    if (currentRightBack != TargetRightBack)
    {
          if (currentRightBack < TargetRightBack)
          {
                currentRightBack = currentRightBack + MotorIncrement;
                //Check if motor has surpassed target value...
                if (currentRightBack > TargetRightBack)
                {
                      currentRightBack = TargetRightBack;
                }
          }
          else
          {
                currentRightBack = currentRightBack - MotorIncrement;
                if (currentRightBack < TargetRightBack)
                {
                      currentRightBack = TargetRightBack;
                }
          }
    }

    // for left back motor
    if (currentLeftBack != TargetLeftBack)
    {
          if (currentLeftBack < TargetLeftBack)
          {
                currentLeftBack = currentLeftBack + MotorIncrement;
                //Check if motor has surpassed target value...
                if (currentLeftBack > TargetLeftBack)
                {
                      currentLeftBack = TargetLeftBack;
                }
          }
          else
          {
                currentLeftBack = currentLeftBack - MotorIncrement;
                if (currentLeftBack < TargetLeftBack)
                {
                      currentLeftBack = TargetLeftBack;
                }
          }
    }
motor[RightBack] = currentRightBack;
    //wait1Msec(1);
    motor[LeftFront] = currentLeftFront;
    //wait1Msec(1);
    motor[RightFront] = currentRightFront;
    //wait1Msec(1);
    motor[LeftBack] = currentLeftBack;
```

```
      //wait1Msec(1);
}

bool SensorCheck (char cmd)
{
      if(cmd == 'f' && SensorValue [FrontSensor] < DISTANCE)
      {
            return false;
      }
      else if(cmd == 'r' && SensorValue [RightSensor] < DISTANCE)
      {
            return false;
      }
      else if(cmd == 'l' && SensorValue [LeftSensor] < DISTANCE)
      {
            return false;
      }
      return true;
}
task main()
{
  motor[LeftBack] = 0;
  motor[RightBack] = 0;
  motor[LeftFront] = 0;
  motor[RightFront] = 0;
  TargetRightFront = 0;
  TargetLeftFront = 0;
  TargetRightBack = 0;
  TargetLeftBack = 0;
  currentRightFront = 0;
  currentLeftFront = 0;
  currentRightBack = 0;
  currentLeftBack = 0;
  watchDog = 0;
  char cmd;
  bNxtLCDStatusDisplay = true;
  //wait1Msec(2000); // Give time to start the program at the far end as well
  int time, now, timePlus30;
  ClearTimer(T1);
  time = time1[T1];
  // watit until a bluetooth connection is established
  while(!checkBTLinkConnected())
    ;
  while(1)
  {
      timePlus30 = time + 30;
    do
    {
        now = time1[T1];
    }while(timePlus30 > now && now > 0);
      ClearTimer(T1);
      time = time1[T1];
        bluetoothCheck(cmd);
        if(!SensorCheck(cmd))
        {
                  SetNewTargetSpeeds('e', 0);
                  PlaySound(soundShortBlip);
```

```
                    continue;
                }
                updateSpeed();
            }
            return;
        }
```

Android Code

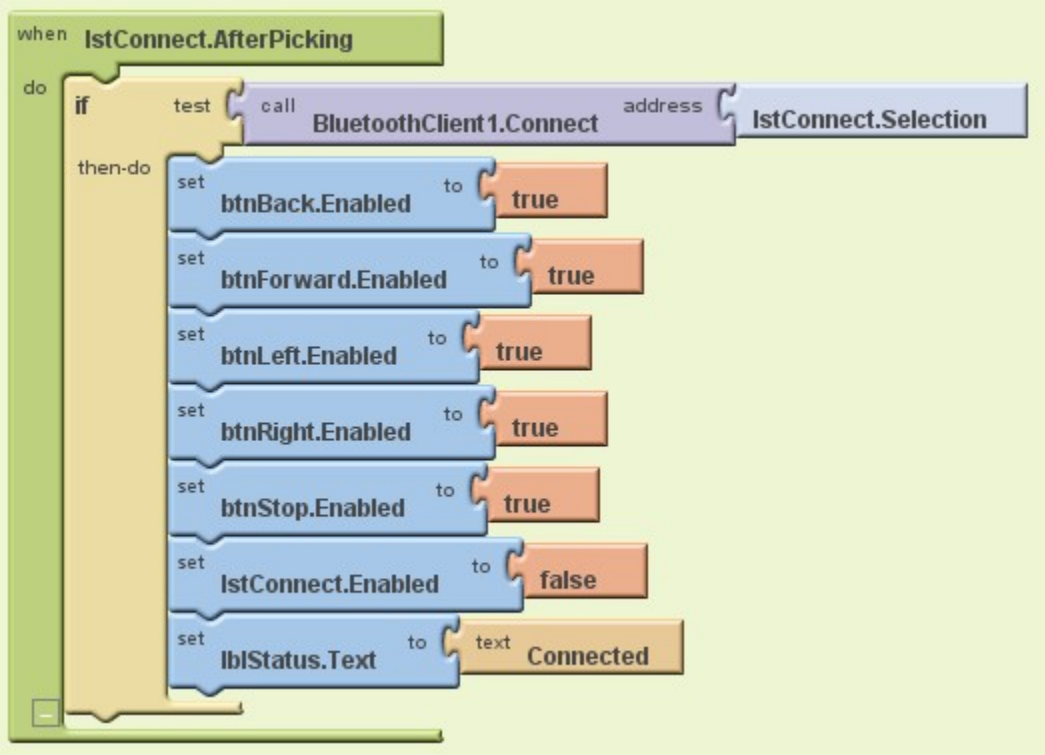


Figure 49: Android code to connect NXT for

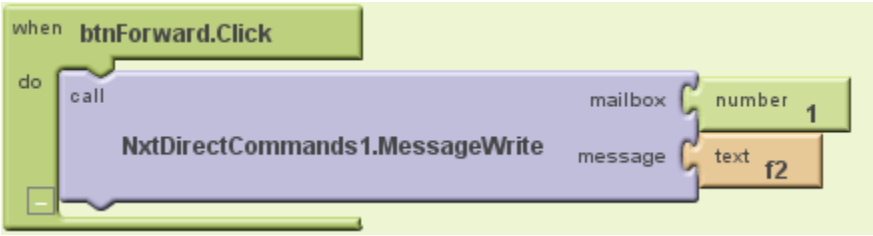


Figure 50: Android code to send forward command with speed 50

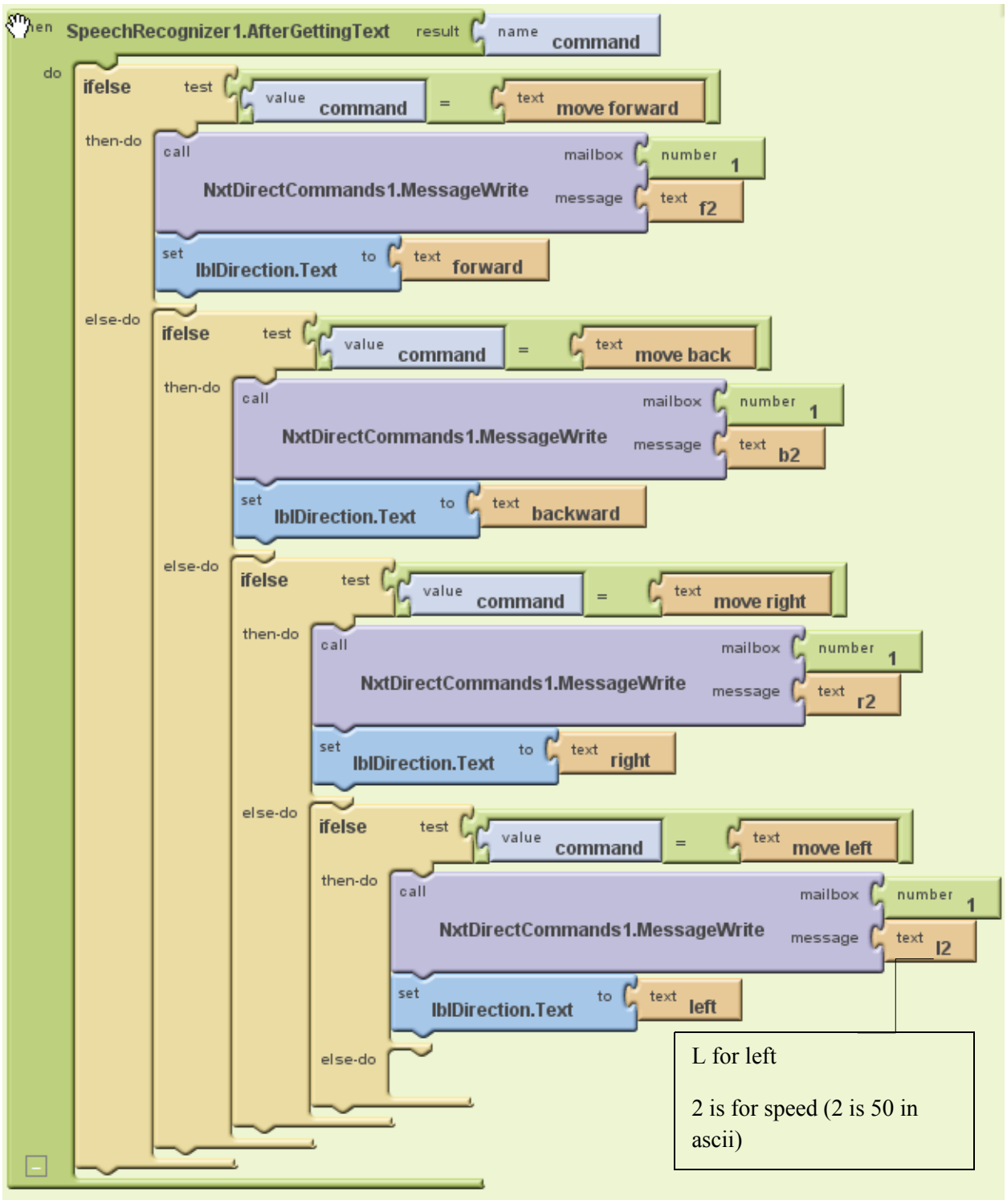


Figure 51: Android Code for speech recognition

CUDA sample

```
// OpenCVGpu.cpp : main project file.
// written by: Gehad Shaat
#include "stdafx.h"
#include <OpenCV2/core/core.hpp>
#include <OpenCV2/features2d/features2d.hpp>
#include <OpenCV2/highgui/highgui.hpp>
#include <OpenCV2/GPU/GPU.hpp>
using namespace System;

using namespace cv;
using namespace cv::GPU;
static bool USE_VIDEO = true;
int main(array<System::String ^> ^args)
{
	if(cv::GPU::getCudaEnabledDeviceCount() == 0)
		return -1;
	cv::VideoCapture capture(0);
	if(!capture.isOpened()) {
		 USE_VIDEO = false;
	}
	cv::GPU::DeviceInfo devInfo;
	//variables
	GpuMat GpuSrc(0,0,0);
	GpuMat GpuDst(0,0,0);
	GpuMat GpuTmplt(0,0,0);
	Mat src;
	Mat dst;
	Mat tmplt;
	tmplt = imread("template.jpg", CV_LOAD_IMAGE_GRAYSCALE);
	GpuTmplt.upload(tmplt);
	if(!USE_VIDEO) {
		src = imread("file.jpg", CV_LOAD_IMAGE_GRAYSCALE);
		GpuSrc.upload(src);
	}
	while(1)
	{
		if(USE_VIDEO) {
			capture.read(dst);
			cvtColor(dst, src,CV_RGB2GRAY);
			GpuSrc.upload(src); //upload image to GPU
		}
		double cpuT = (double)cvGetTickCount();
		cv::matchTemplate(src,tmplt ,dst, CV_TM_CCORR_NORMED);
		cv::imshow("Src CPU image", src);
		cv::imshow("dst CPU image", dst);
		cpuT = (double)cvGetTickCount() - cpuT;
		printf( "cpu time = %g ms\n", cpuT/
((double)cvGetTickFrequency()*1000.) );
		cv::waitKey(1);
		// GPU threshold
		double GPUT = (double)cvGetTickCount();
		cv::GPU::matchTemplate(GpuSrc,GpuTmplt ,GpuDst,
CV TM CCORR NORMED);
		cv::imshow("Src GPU image", GpuSrc);
		cv::imshow("dst GPU image", GpuDst);
```

```
            GPUT = (double)cvGetTickCount() - GPUT;
            printf( "GPU time = %g ms\n", GPUT/
((double)cvGetTickFrequency()*1000.) );
            cv::waitKey(1);
            if(!USE_VIDEO) {
                  break;
            }
      }
      cv::waitKey();
      return 0;
}
```

TBB code sample

parallelizing two functions described in CUDA sample code

```
// written by: Gehad Shaat
#include <iostream>
#include <list>
#include <tbb/task.h>
#include <tbb/task_group.h>
#include <stdlib.h>
#include <OpenCV2/core/core.hpp>
#include <OpenCV2/features2d/features2d.hpp>
#include <OpenCV2/highgui/highgui.hpp>
#include <OpenCV2/GPU/GPU.hpp>

using namespace cv;
using namespace cv::GPU;
static bool USE_VIDEO = true;
using namespace tbb;

//variables
static GpuMat GpuSrc(0,0,0);
static GpuMat GpuDst(0,0,0);
static GpuMat GpuTmplt(0,0,0);
static Mat src;
static Mat dst;
static Mat tmplt;

//need to create class for CPU to be used with TBB
class CPUmatch
{
public:
      void operator()()
      {
            double cpuT = (double)cvGetTickCount();
            cv::matchTemplate(src, tmplt,dst,CV_TM_CCORR_NORMED);
            cv::imshow("Src CPU image", src);
            cv::imshow("dst CPU image", dst);
            cpuT = (double)cvGetTickCount() - cpuT;
            printf( "cpu time = %g ms\n", cpuT/
((double)cvGetTickFrequency()*1000.) );
            cv::waitKey(1);
      }
```

```
};

//need to create class for GPU function to be used with TBB
class GPUmatch
{
public:
	void operator()()
	{
		// GPU threshold
		double GPUT = (double)cvGetTickCount();
		cv::GPU::matchTemplate(GpuSrc,GpuTmplt,
GpuDst,CV TM_CCORR_NORMED);
		cv::imshow("Src GPU image", GpuSrc);
		cv::imshow("dst GPU image", GpuDst);
		GPUT = (double)cvGetTickCount() - GPUT;
		printf( "GPU time = %g ms\n", GPUT/
((double)cvGetTickFrequency()*1000.) );
		cv::waitKey(1);
		GpuDst.release();
		GpuSrc.release();
	}
};

int main(int argc, char** argv)
{
	if(cv::GPU::getCudaEnabledDeviceCount() == 0)
		return -1;
	cv::VideoCapture capture(0);
	if(!capture.isOpened()) {
		USE_VIDEO = false;
	}
	cv::GPU::DeviceInfo devInfo;
	//variables
	tmplt = imread("template.jpg", CV_LOAD_IMAGE_GRAYSCALE);
	GpuTmplt.upload(tmplt);
	if(!USE_VIDEO) {
		src = imread("file.jpg", CV_LOAD_IMAGE_GRAYSCALE);
		GpuSrc.upload(src);
	}
	while(1)
	{
		if(USE_VIDEO) {
			capture.read(dst);
			cvtColor(dst, src,CV_RGB2GRAY);

			GpuSrc.upload(src);
		}
		task_group group;
		try
		{
		group.run(CPUmatch());
		group.run(GPUmatch());
		group.wait();
		} catch (...) {
			;
		}
```

```
	}
	return(0);
}
```

Nathen's Original Code for the NXT

```
#pragma config(Hubs,  S1, HTMotor,  HTMotor,  HTMotor,  none)
#pragma config(Sensor, S2,     LeftSensor,
	sensorSONAR)
#pragma config(Sensor, S3,     RightSensor,
sensorSONAR)
#pragma config(Sensor, S4,     FrontSensor,
sensorSONAR)
#pragma config(Motor,  mtr_S1_C1_1,     LeftBack,
	tmotorNormal, PIDControl, reversed, encoder)
#pragma config(Motor,  mtr_S1_C1_2,     RightBack,
tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C2_1,     RightFront,
	tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C2_2,     LeftFront,
tmotorNormal, PIDControl, reversed, encoder)
#pragma config(Motor,  mtr_S1_C3_1,     Neck,
		tmotorNormal, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C3_2,     motorI,
	tmotorNormal, PIDControl, encoder)
//*!!Code automatically generated by 'ROBOTC' configuration wizard
                                     !!*//

void mad ()
{
	motor[Neck]=-05;
	wait1Msec (300);
	motor[Neck]=05;
	wait1Msec (700);
	motor[Neck]=-05;
	wait1Msec (800);
	motor[Neck]=0;
}
void moveforward ()
{
	motor[LeftFront]=50;
	motor[RightFront]=50;
	motor[LeftBack]=50;
	motor[RightBack]=50;
	wait1Msec (3000);
}
void moveright ()
{
	motor[LeftFront]=50;
	motor[RightFront]=-50;
	motor[LeftBack]=50;
	motor[RightBack]=-50;
	wait1Msec (2000);
}
void moveleft ()
```

```
{
	motor[LeftFront]=-50;
	motor[RightFront]=50;
	motor[LeftBack]=-50;
	motor[RightBack]=50;
	wait1Msec (2000);
}
void moveback ()
{
	motor[LeftFront]=-50;
	motor[RightFront]=-50;
	motor[LeftBack]=-50;
	motor[RightBack]=-50;
	wait1Msec (3000);
}
void fastleft ()
{
	motor[LeftFront]=-50;
	motor[RightFront]=50;
	motor[LeftBack]=50;
	motor[RightBack]=-50;
	wait1Msec (3000);
}

void fastright ()
{
	motor[LeftFront]=50;
	motor[RightFront]=-50;
	motor[LeftBack]=-50;
	motor[RightBack]=50;
	wait1Msec (3000);
}

void scan ()
{
	int scan1;
	int scan2;
	int scan3;
	int scan4;
	int scan5;
	int counter=0;
	scan1 = SensorValue [LeftSensor];
	scan3 = SensorValue [RightSensor];
	wait1Msec (500);
	scan2 = SensorValue [LeftSensor];
	scan4 = SensorValue [RightSensor];
	int result1=scan1-scan2;
	int result2=scan4-scan3;
	scan5 = SensorValue [FrontSensor];
	if (scan5 <29)
	{
		if (result1 > 0)
		{
			moveleft ();
		}
	}
	if (scan5 <29)
```

```
	{
		if (result2 < 0)
		{
			moveright ();
		}
	}
	if (scan4 <29)
	{
		if (scan5 < 29)
		{
			moveleft ();
		}
	}
	 if (scan2 <29)
	{
		if (scan5 < 29)
		{
			moveright ();
		}
	}
	if ((scan5 <29) && (scan2 <29) && (scan5 <29))
	{
		mad ();
		moveleft ();
		moveleft ();
		fastleft ();
		while (counter == 0)
		{
			if (scan5 < 29)
			{
				fastleft (  );
			}
			else
			{
				counter=(counter+1);
			}
		}
	}
}
//use 'void varname () {}' for creating variables
// varname () recalls variable varname in the program
// int name=0; sets integer name to zero

task main ();
{
	wait1Msec (3000);
	while (1==1)
	{
		scan ();
		moveforward ();
	}
}
```