Brutus vs. Albert

Hubert Ho

EEL 5666
Dr. Arroyo
Final Report
December 10, 2002

Table of Contents

Abstract……………………………………………………… 3
Executive Summary ………………………………………………. 4
Introduction……………………………………………….............. 5
Integrated System…………………………………………….. 7
Mobile Platform………………………………………………… 9
Actuation……………………………………………………… 11
Sensors……………………………………………………….. 13
    Bump/Switch Sensors……………………………………….. 13
    Sharp Distance Sensor…………………………………….… 14
    CMUcam (Special Sensor)…………………………….……….. 15
        The Problem……………………………………….. 15
        Test Images………………………………………… 17
        The Solution……………………………………… 19
        Implementation…………………………………… 21
        Analysis…………………………………. ……. 22
Addition Hardware……………………………………………… 23
    Solenoid…………………………………………….………. 23
    Voice Playback……………………………………….……. 24
    Speaker…………………………………………………. 26
    Power……………………………………………….……… 27
    Marker……………………………………………….…….. 28
Behaviors……………………………………………………… 29
Experimental Layout and Results……………………………… 32
Conclusion………………………………………………….…….. 33
Prediction………………………………………………………… 36
Documentation………………………………………………….… 37
Appendices…………………………………………………….……. 38
    Code……………………………………………………. 39

Abstract

Brutus is a robot project, named after the "Best College Mascot" – Brutus Buckeye of The Ohio State University. Brutus is able to determine between seeing either the mascot of OSU, Brutus Buckeye, or the mascot of University of Florida, Albert Alligator. After determining which mascot is detected, the robot will perform "Script Ohio" for OSU, and "Gators" for UF, while playing the fight song of the respected school.

Executive Summary

As a current student of University of Florida, and an alumni of The Ohio State University, football plays a huge role during Fall quarter/semester. By implementing both of these schools together, this robot project will integrate both teams into one project, with a performance for both schools to be proud of.

The Ohio State University's marching band is one of the best in the country, and a proud tradition of this powerhouse school is "Script Ohio". The Florida marching band also exhibits a similar performance with "Gators", and by playing the fight song of the respect school, only adds to the performance to make it stronger and better.

The microprocessor board used is an Atmel Mega163 board, which is used to integrate and implement the whole project. The platform of the robot is built out of wood and stable enough to handle all of the different aspects required for the robot. Many sensor are used on this robot, but the "special" sensor is a CMUcam, which is a low-cost camera used for robot applications. The code for the program is written in C code and downloaded onto the chip to be tested and ran.

Tying everything together into one package, proved to be very time consuming and difficult, but achievable. Being able to exhibit the correct behaviors necessary is a difficult task, but one to be proud of when done correctly.

Introduction

As an alumni of The Ohio State University Buckeyes, this fall has been an amazing experience. Since graduating from OSU last spring, and making the decision to continue my education here at University of Florida, relocating from one football school to another football school definitely has its benefits. Both of these highly respected schools and programs have had successful seasons. Ohio State has answered all questions by finished its season undefeated (13-0), and deserving our chances to play for the National Championship on January 3rd in Tempe, Arizona, against University of Miami. Florida has also had a successful season under a first year coach, and has their mindset on the Outback Bowl in Tampa vs. University of Michigan.

By incorporating the football atmosphere of both of these schools, I am able to create my robot: Brutus (named after Brutus Buckeye, the mascot of The Ohio State University), which is my project for IMDL (Intelligent Machine Design Laboratory), that will be able to detect either the OSU mascot, Brutus Buckeye, or the UF mascot, Albert Alligator. After the detection of either of these mascots, the robot's microprocessor board (Atmel Mega163), will be able to determine which of these mascots is being seen. After detection, the robot will turn around, and perform either "Script Ohio" (Figure 1) if Brutus (Figure 2) is seen, or "Gators" (Figure 4) if Albert (Figure 3) is detected, while playing the fight song of the respected school.

Figure 1 – "Script Ohio"

Figure 2 – Brutus Mascot **Figure 3 – Albert Mascot**

Figure 4 – "Gators"

Integrated System

The Atmel Mega163 Microprocessor Chip was included with the purchase of the Progressive MegaAVR-Dev development board (Figure 5). This board turned out to be very helpful and useful for understanding the microprocessor chip, which was aided very nicely by the included documentation. The fact that everything was laid out nicely and readily available was a huge plus, especially for someone who does not have a vast understanding and background with these kinds of chips.

The board assisted nicely in having input and output pins for sensors and such, along with the power connections, serial connections, a reset button, and LED's. Most of the hardware became very easy to integrate with the development board. The 2 servo motors (used for moving the wheels of the robot) were connected to PortD pins 4 and 5, which were the PWM (Pulse Width Modulation) timer functions. The sound chip (used to play the fight songs of the schools) were connected to PortB pins 0,1, and 2. PortA was used to connect the Sharp IR distance sensor (measures distances in front of the robot), and a Bump sensor (used to trigger the start of the robot). The solenoid (used to move the marker up and down) used PortC pin 1. The CMUcam, used to determine the mascot in front of the robot, uses the serial port to communicate its results.

With these many sensors, and different input/output values, all integrated into one microprocessor board of this robot platform, programming the chip to work correctly became a humongous task. The programming software that is available for the Progressive Development board, is the AVR Studio, which programs in the C language. This language by itself is difficult to understand, but luckily there are test programs and also other examples by other students for us to browse and to help us understand. After

spending much time testing and understanding the program, I was finally able to create my own programs and integrate the test examples into my program to result in the outcome I desired.

“Script Ohio” and “Gators” were the outcome of much testing and coding for the correct output and results presented. There were many other factors that made the performances difficult, which include the platform itself, the wheels used, the servos, and also the ground the robot is on.

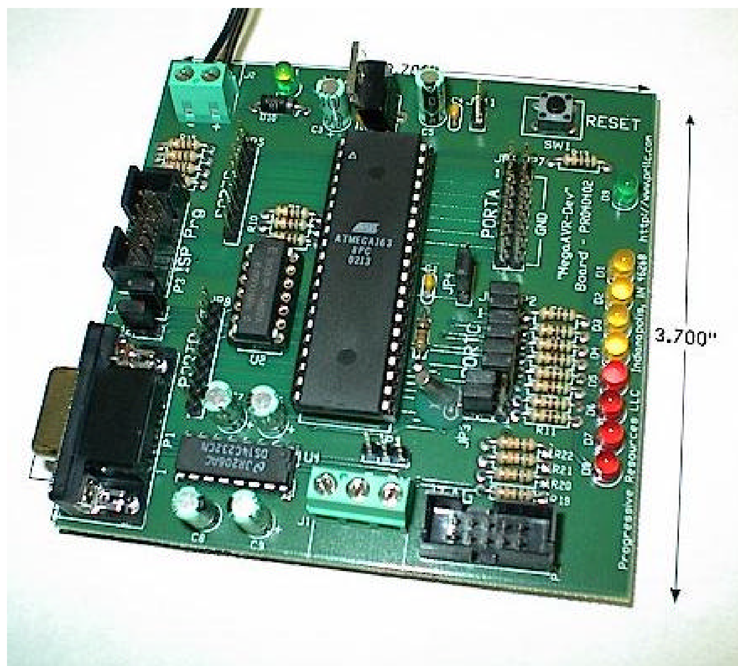

Figure 5 – Atmel Mega163

Mobile Platform

The body of my robot was based on the many TJs (Talrik Juniors) that the MIL lab has. Although the robot will need to perform difficult tasks, the platform of the TJ appeared to be sufficient enough for my requirements because of the fact that not many mechanical functions needed to be factored into the design. I knew that the TJ design would be too small, and so I created a design that was a little larger and would be able to provide me more functionality and possibilities.

The design of the many pieces used in creating the robot's platform were designed using AutoCAD2002 (Figure 6), which proved to be somewhat of a difficult task for someone who is not a Mechanical Engineer, but easy enough to learn and use. After designing the pieces, the T-Tech machine was then programmed to cut out the wood, based on the design created in AutoCAD. Something that I had to keep in mind was that the wood was 1/8 inch thick, and thus my design had to factor that size in, and the side pieces had to allow enough room for the other pieces to fit into correctly.

After cutting out the pieces, I sanded down the sides and glued the pieces together to form the body. The body was large enough to place my microprocessor board on, a place in the back for my battery pack, enough room for my servos to be mounted, and the top had room for both my camera and a helmet to be attached. One thing that I did need to add to this design was an arm on the bottom for controlling the solenoid, which was responsible for my marker to draw the words "Ohio" and "Gators". The was a huge headache, but finally achieved successfully.

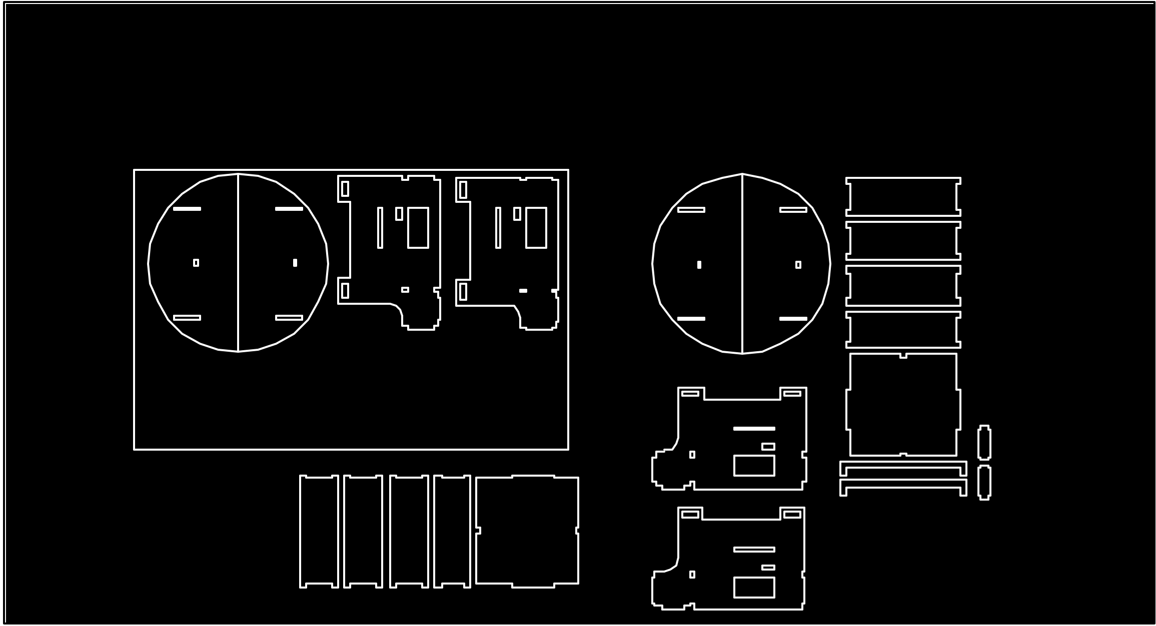

Figure 6 – AUTOCAD design

Actuation

My robot used two hacked servo motors (as the source of locomotion), which seemed to give me some trouble because of the fact that the hacked servos were not precise enough as my application would have liked. These servos were easier to integrate into this system and with my microprocessor board, as compared to motors, because the chip already had pins ready for PWM (Pulse Width Modulation) that can handle the servo movements. I had considered using stepper motors, but turned away because motors required special stepper motor drivers and they cost more then I would have liked. I had hoped that the servos would be enough to handle my application, and they proved to be successful, but not perfect.

By hacking the servos and achieving continuous movement, this allowed me to mount one servo on the left and one on the right side of the robot. By spinning one servo one way, and the other servo the opposite way, I was able to have the robot move straight. The values needed by the servos to move straight were difficult and required many tests before being found. Similarly, many tests were needed in order to have the robot move backwards, turn right, and also turn left. These values were only found after careful testing and trials.

The servos were Balsa Products BP148N 2BB Standard-torque ball-bearing servo motors, from MarkIII (Figure 7). The wheels used were injection molded wheels (Figure 8), also bought from MarkIII, which fit perfectly on the servos. The wheels were hard plastic wheels, but by attaching the included rubber bands, the wheels thus had friction and a way to stick to the ground and move the robot.

While each of the servos make up 2 points that the robot touches the ground, another point is needed in order to make the robot stable. This last point is made up of a plastic furniture glide piece, located in the rear of the robot, which sticks into the bottom of the robot and allows it to glide on the surface of the floor. These three points stabilized the robot enough so that it can move accurately and perform the necessary actions.

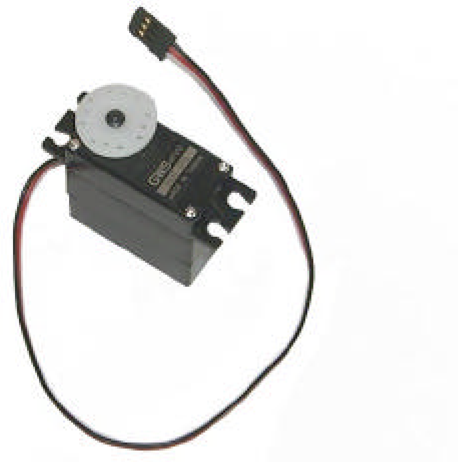

Figure 7 – Servo Motors

Figure 8 – Wheels

Sensors

Bump/Switch Sensors

After the robot has been turned on, it will be inactive and wait until a bump switch is pressed by the user. This will then trigger the robot to begin the performance. The main function and purpose of this is so that the robot will be in control of the user and hopefully not somehow break itself by falling off a table, or running somewhere it could possibly be damaged. I found this sensor very easy to use and also very important in order for the robot to begin correctly.

The Bump Sensor that I implemented was a simple snap-action switch, ordered from Jameco (Figure 9). The actuator used to make the contact was a hinge lever, and while the switch is open, a low value is read in, until the sensor has a "hit", which makes the value then high. I only used one of these sensors because of the specifications of my robot does not need to worry about running into anything.

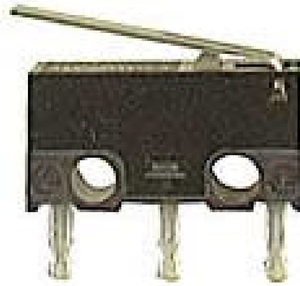

Figure 9 – Bump Sensor

Sharp Distance Sensors

By using the Sharp Distance Sensor (Figure 10), the robot will be able to tell if there is actually something in front of it. If we were to just rely on the photo images and pattern extraction techniques, the robot may incorrectly qualify something as being either mascot, which in reality is actually wrong. By using these sharp sensors, the robot will be able to determine its exact distance to the object.

The Sharp Distance Sensor that I have chosen is the Sharp GP2D12 Infrared Ranger. I am using one of these sensors, which is mounted on the front center of the robot, pointing straight forward. The sensor then returns the value to the Atmel Progressive Board for processing.

The distance is measured by actually having one side being a light emitter, and the other side being a light detector, where the light emitter sends a light beam, which is reflected off an object, and returned back to the light detector. The distance from the sensor to the object is proportional the time the reflection is detected back. This means that when the object is close to the sensor, the value sent to the board is higher then when it is far away, where the value is lower.

It turned out that I only wanted to find the distance of about a foot away from the object, and so after many tests, the correct value of 24 was the value looked at by the sensor. Once a value of 24 was returned to the microprocessor, the robot then knew to stop looking for mascot and a picture could then be taken by the CMUcam.

Figure 10 – Sharp Distance IR Sensor

CMUcam (Special Sensor)

The Problem

The Brutus robot is using a CMUcam Vision Board (Figure 12) camera to take images of what it is facing. The webpage and description of the CMUcam can be found at its website: www-2.cs.cmu.edu/~cmucam. This camera is a "new low-cost, low-power sensor for mobile robots." This cost of this camera is $109 through Seattle Robotics, and can be found at www.seattlerobotics.com/cmuinfo.htm. This camera uses a serial port, which makes it easy to connect to both the robot's microprocessor board, or a computer's serial port.

While being connected to a computer, the camera's package also includes two demo programs which allow for the user to test the camera and see the images. In addition to seeing the images of the camera, the demo programs also allow the user to test the many different functions of the camera, which include the following: setting the camera's internal register values, dumping frames, delaying packets transmitted, getting the mean color values, getting the current version of software, controlling the tracking light of the camera, and several others. Other characteristics of the CMUcam is that it can track color blobs at 17 frames per second, gather mean color and variance data, transfer real-time binary bitmap images, dump raw images, and it has a resolution of 80x143 pixels.

Upon trying to take images in different lighting conditions, it turns out that the camera does not have a very good IR filter lens, which limits the conditions that the robot will be able to work in. In areas with high IR, such as from light bulbs, the images appear very dark and red, which makes determining the true color of objects very hard.

Another problem with the images of the two mascots in the pictures became the fact that if the images would have very different values depending on the distance of the camera to the image. Another problem arose being the fact that if the background of the mascot is random, then the color pixels of the images would very greatly from one set of pictures to another. These problems would be dealt with by limiting the conditions of the camera's images.

After connecting the camera to the microprocessor board, and dumping the images, the microprocessor board will have to determine if the image contains either Brutus (OSU's mascot) or Albert (UF's mascot). This became the main problem and focus for my Pattern Recognition Project. The problem of being able to interface the CMUcam with the Atmel Mega163 (Figure 11) microprocessor chip was very time consuming and full of headaches. Luckily because of the fact that the Atmel processor board suggests the use of AVR-Studios, which programs in C, to program the board, it became easier to understand how to operate the CMUcam with the microprocessor board.

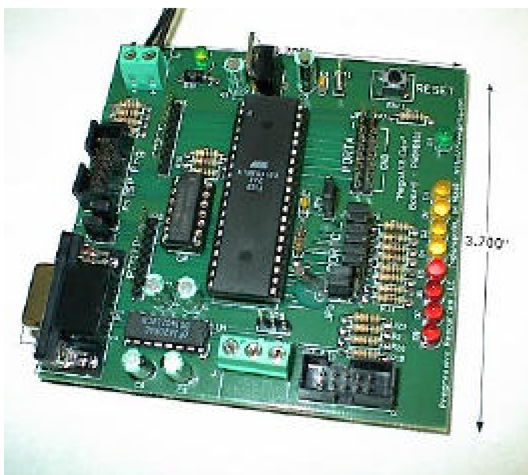

Figure 11 – Atmel Mega163 Microprocessor

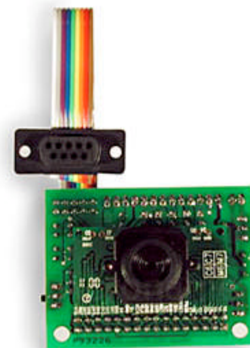

Figure 12 – CMUcam

Test Images

By implementing the two separate demo programs for the CMUcam, I was able to test the camera for different results by sending the camera different commands through the serial port. In using different commands and setups, I was able to see different results in the images captured and understand what the optimal setup for the camera would be.

The best image, as far the size of the mascot as compared to the whole image, would be one where the mascot is seen from the bottom of the image to the top. This turned out to be when the robot is about a foot away from either mascot. By restricting the camera to only capture the images while being a foot away from the mascot, we are increasing the chances of having a consistent image to be classified. This actually turned out to be very important and very necessary as testing increased and became more difficult.

With the image, another problem that arose was the fact that the background behind the mascot changes depending where the mascot is located. With the background changing, the different colors and objects in the whole image would make the mascots very hard to classify and separate. After much agony and turmoil, the decision was made to make the background consistent and not changing. I was able to do this by putting up a white bed sheet behind the mascot, thus making the background always white and only the mascots' colors would be processed.

In addition to the backgrounds not being consistent, another problem arose, which is a very common and always eminent problem, of lighting. Depending on the lighting of the atmosphere of room that the testing is done, the images would vary greatly. The problem of the CMUcam not having a very good IR filter plays a huge part here, in that

light bulbs result in images that are very dark, red, and hard to distinguish objects and colors (Figure 13). The solution to this problem was using a florescent light, which I hold over the robot, facing the mascots. Not only did this solution fix the problem of the redness, the florescent lights also seemed to brighten up the mascots' colors, making them very distinguishable and easy to pick out (Figure 14).

These changes appeared to be good solutions to the image problem, in that the images became very consistent. By having consistent images captured by the CMUcam, this allows for testing and classifying to be easier and more accurate. On the bad side, the robot is restricted to only these conditions stated above. I feel that although the robot is restricted, this is still a difficult application to make work and this is a very good start to answer this problem.

Figure 13 – no IR filter (regular light bulb)

Figure 14 – IR filter (florescent light)

The Solution

By taking the images captured from the CMUcam, which are saved on the computer, I am able to use software to manipulate the data and find the best way to classify the two mascots. The classifier that I chose for separating the two mascots deals with color because each of the mascots have definite colors that separate one from the other. The Brutus Buckeye mascot has mostly red, tan, and dark brown materials, while Albert Alligator is made up of mostly orange, dark green, and light green. In applying the techniques from homework 2, problem number 3, instead of looking at each pixel as a 3 dimensional color scheme of Red, Green, and Blue (RGB), and because the color blue is very small, we can choose to leave blue out of our calculations. By having a 2 Dimensional vector, the amount of information needed to be processed is reduced greatly.

By taking these 2D color vectors (Figures 15 and 16), and analyzing the images with Mathematica, we are able to create a good classifier, with graphs and Gaussian distributions for the different cases. With these classifiers, we are able to implement separations and understand what the microprocessor board will have to do in order to process the different images.

The technique that I initially wanted to implement was to read in each pixel of an image and classify it as either being: Brutus, Albert, or neither. Where after all of the pixels in an image have been evaluated, the board should be able to determine which mascot is seen by the camera. While this would have been a very nice classifier, I discovered that the CMUcam already has a function that would be able to "Get the Mean color value in the current image" (which means the RGB pixel averages). With this

function already available, it would be much easier to just test the mean RGB values and discover a good way to classify the two mascots in this way.

By taking the best images of the mascots captured from the CMUcam, and importing them into Mathematica, I was able to find the mean pixel RGB values. These values are as follows: for Brutus Buckeye, the mean values were:

{159.826, 115.89, 40.8258} {Red, Green, Blue}.

The values for Albert Alligator were as follows:

{149.82, 119.045, 41.3058} {Red, Green, Blue}.

With interpreting these results, it can be found that the Green and Blue averages were very similar, and possibly not a very reliable classifier. On the other hand, Red stands out to be about a 10 (out of 255) point difference, which could very well be the difference needed to determine between the two objects. By choosing the average value between 159.826(Brutus) and 149.82(Albert) of 155, we have now found the classifying threshold to be implemented into the program for the robot's microprocessor board.

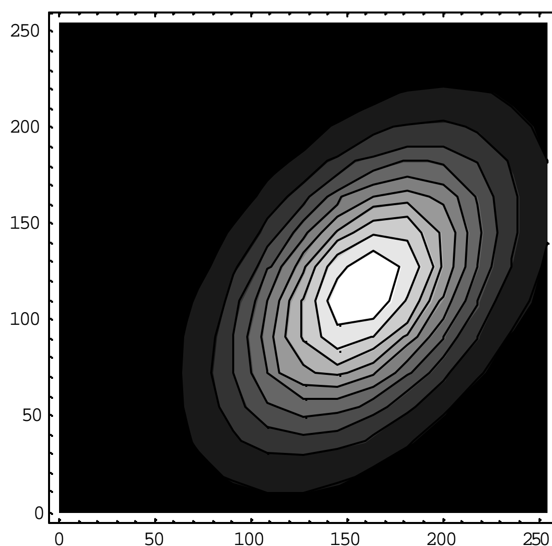

Figure 15 – RG graph of Brutus

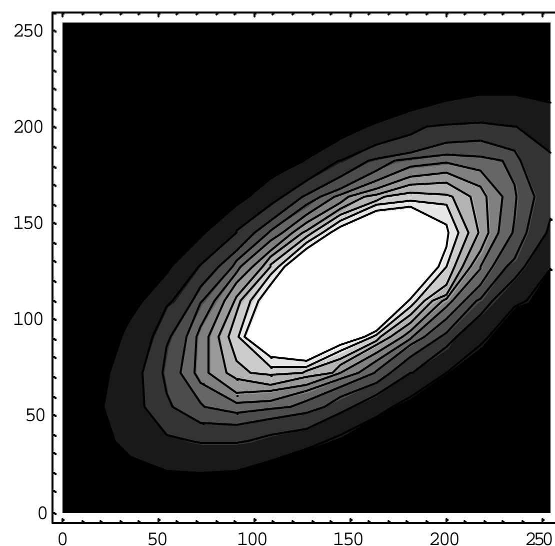

Figure 16 – RG graph of Albert

Implementation

Connecting the CMUcam to the Atmel microprocessor board proved to be very tricky and difficult to implement because of the fact that the camera did not include very detailed instructions on how to handle this topic. The camera has two options – going through the level-shifted serial port, or connecting using the TTL serial port. I found using the level-shifted 9-pin serial port connector easier to implement and configure. The connection was actually made by only using the GND, TX, and RX from the camera to the board.

The code for the camera is as follows:

```
{
        PRINT("RS");                         //resets the camera
        EOL();                               //end of line
        PRINT("PM 1");                       //polls one line at a time
        EOL();
        PRINT("CR 19 32 16 140");            //sets auto-gain and exposure of 140
        EOL();
        PRINT("RM 3");                       //sets raw input and output
        EOL();
        PRINT("GM");                         //gets mean color values
}
```

This code basically transmits the mean color values through the serial port to the microprocessor board. After reading in the mean red value, the microprocessor compares it to the threshold previously determined.

If the value read is less then the classifying threshold of 155, then the mascot is Albert, and if the value is greater then 155, then the mascot is Brutus. This then triggers the performance of the respected mascot. Although this process seems to be fairly easy, testing the values and finally receiving the correct values became a very tough challenge. The difficulty was that if the camera is connected to the microprocessor board, then it can

not be connected simultaneously to the computer, which makes it very difficult to debug and know what values are being read. This problem was finally resolved by having the board send the commands to the camera, receive the values back, and then wait to be triggered to send the values to the computer. Meanwhile, the connections are removed to the camera, and connected to the computer, and finally sending the values to the computer's Hyperterminal. The values can then finally be seen by the user and determined if they are correct or not.

Analysis

While using the mean color values found by the CMUcam to determine threshold differences is not a very difficult application, the fact that I was able to incorporate the Gaussian distributions and mean value techniques covered during class into an actual robot project, makes it a special condition. Mathematica was used to see actual statistical 3D and 2D models for both images of Brutus and Albert. From these contour plot models, we are able to see the differences of the images and know that there is actually a way of classifying one object from another.

From the same Mathematica tests run on multiple images, of different conditions, we are able to determine the best situation and conditions, which were to have the robot a distance of one foot away from the mascot, have a white background behind the mascot, and to have a florescent light providing a consistent lighting. Ideally, with these conditions stated, and the classifying threshold found, the robot should be able to successfully determine between Brutus and Albert.

Additional Hardware

Solenoid

The Solenoid (Figure 17) that I used is a push-type solenoid that was purchased through Jameco, which was used to control the movement of the marker. The solenoid functions by connecting the two wires to either ground or power, causing the plunger to be pushed through and held in place. When the plunger is pushed down, this causes the arm to be lowered, which controls the marker on the end of the arm. When the solenoid is not connected, the arm is raised by a sponge, pushing the marker off of the ground.

The solenoid circuit was connected by implementing an optoisolating circuit, which separates the solenoid from the rest of the robot. The optoisolator used was a Fairchild Semiconductor Photodarlington Optocoupler H11B1, which waits for the signal from microprocessor board, and then sends the signal to the Power Mosfet MTP75N03HDL. The signal from the microprocessor board completes the circuit, which causes the MOSFET to trigger the solenoid and thus lowering the marker.

This circuit proved to be very difficult to understand and figure out. After breadboarding the circuit several times with several different setups, the answer was finally found. Initially the circuit diagrams I had received included using a resistor on the MOSFET, which I found to be unnecessary and caused the circuit to not work properly. After many tests, and finally removing this resistor, the circuit finally worked, and the marker moved correctly.

Figure 17 – Solenoid

Voice Playback

The ISD25120 (Figure 18) was used to play the fight song of both schools. The chip has the capability to playback 120 seconds worth of input recordings at a frequency of 4kHz. While my application only needed to choose between a total of only two "message" choices, the chip has the capability of playing any number of messages. There are several different ways to record onto the chip, and also to playback the messages from the chip. The easiest way I found was to use the breadboard setup to record onto the chip and then use the microcontroller to control the different input lines of the chip, for playback purposes.

By using the pushbutton setup for the chip (Figure 19), I was able to simply control the recording, playback, and pausing with switches from a breadboard. By removing the switches and using the microcontroller, I had to solder the chip onto a proto-board and wire-wrap the connections together. This was a very difficult and confusing process, but also a very educational and helpful experience also. Each of the input lines for the chip were controlled by the microcontroller, which was a difficult task to implement, but when finally done right, the correct song was finally able to be played.

The way I recorded onto the chip, the OSU fight song was first, and then the UF fight song second. In order to play the OSU song, I just had to turn on the play line on the board to true. In order to play the UF song, I had to set the mode to a different addressing mode, in order to advance to the second song, and return back to the normal mode, and then play the song. This process took a considerable amount of testing, but I was finally able to have it work correctly.

Figure 18 – Voice Playback Chip

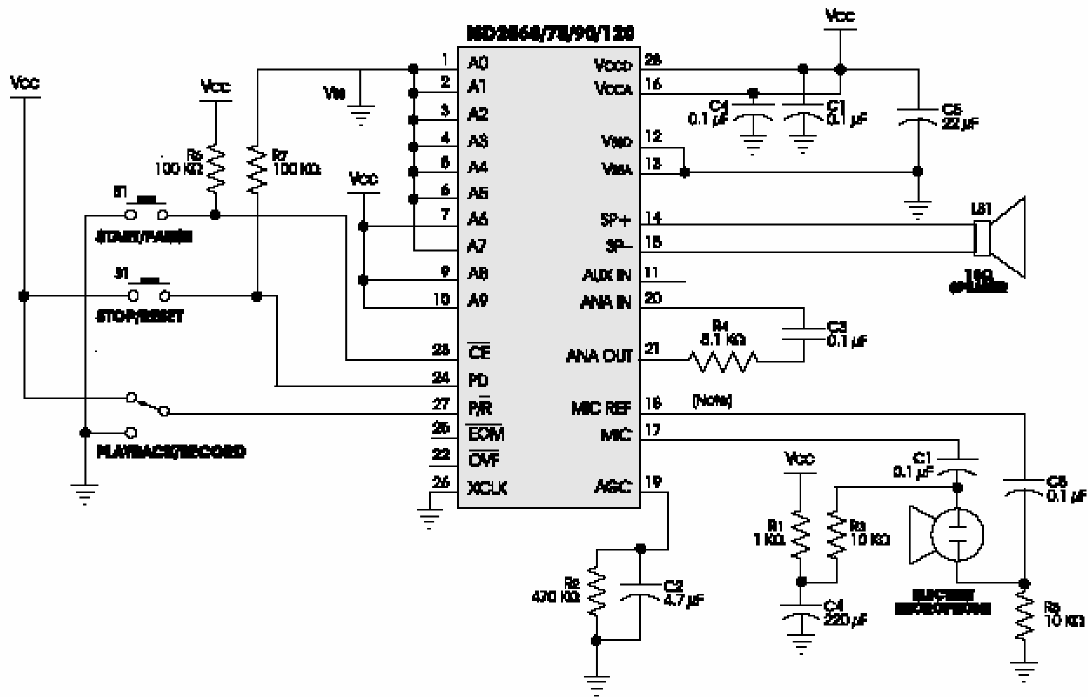

Figure 2-39: ISD2500 Application Example— Push-Button

Figure 19 – Chip Wiring

Speaker

While the speaker (Figure 20), for the voice playback chip, was a standard 8 ohm speaker, I had problems trying to get it to amplify the sound. The servos turned out to be very loud and difficult for the speaker to overcome by itself. I tried to integrate an amplifier chip into the speaker input, but the results were not very positive. I found that the speaker just by itself was louder then with the amplifier. This was somewhat annoying, but in the end I was able to amplify the sound without the chip.

My solution to the speaker's sound being to soft was to use a Styrofoam cup to amplify the sound. I cut out the bottom of the cup, and attached the speaker to the open, bottom end. The sound was able to vibrate and while being directed out of the cup, became loud enough to overcome the sound of the servos running.

Figure 20 - Speaker

Power

For the source of power to operate the robot, I chose to use eight AA Nickel Metal-Hydride batteries (Figure 21). This total gave out around 10.5V, fully charged. While the microprocessor board has its own voltage regulator, the power should be fine as long as it was above 7V. The board then also was able to have its own power output of 5V, which was used to power the bump switch, Sharp IR sensor, and the voice playback chip.

The direct power of the batteries were also used to power the CMUcam, solenoid, and the servo motors. However, before the power could be used by the servos, I used a LM7805 Voltage Regulator in order to create a consistent 5V to each of the servos. This became very helpful and necessary in order to have a constant condition for the motors. The power going to the CMUcam does not necessarily have to be very high, but the CMUcam board can handle it because it has its own voltage regulator on its board. The solenoid runs best on 12V, but by giving it 10.5V, it was able to operate well enough.

Figure 21 – Ni-MH Batteries

Marker

The marker I ended up using was a medium point Boone dry-erase marker. This was the marker I chose because of the fact that it was thin enough to fit in the robot platform. When the robot was finally put together I had problems with the marker not being able to touch the ground. After testing the solenoid circuit over and over, I finally discovered the fact that the marker was too tall and being interfered with by the wires of the robot.

I spent time altering the length of the marker so that it was shorter and then not be able to touch the wires. This appeared to be a good solution and cause the outcome to be more positive. The problem with using markers is that they dry out very easily, and thus I had to have a constant supply of markers, always being altered and ready to use new ones.

Behaviors

When Brutus starts up, it waits for the bump sensor to be pressed, triggering it to finally its demonstration. The first thing that happens during this demonstration is that it looks for a mascot in front of it. It does this by using the Sharp IR Distance Sensor, mounted on the front of the robot. The robot will slowly spin to its right until something is sensed within two feet in front of the robot. I have limited the world that the robot is placed in, so that anytime it senses anything within two feet, it knows that it's a mascot.

After detecting the mascot, Brutus then tries to position itself about a foot away from the object. By placing itself at a distance of a foot away, this allows for the CMUcam to capture a good image of the mascot in order to determine which mascot it is- either Brutus or Albert. The robot then takes the picture with the CMUcam and determines which mascot it is by taking the average pixel values, found by the camera.

If the object is determined to be Brutus, the robot then turns to the right and beings writing out "Script Ohio" and playing the Buckeye Fight Song. In the same manner, if Albert is seen, the robot backs up and beings to write out "Gators" and playing the Florida Fight Song. The solenoid is triggered to activate and push down the marker in order for the path of the robot to be traced and left behind so that the two words can be seen.

The performances of "Script Ohio" (Figures 22) and "Gators" (Figures 23) were a huge headache because of the fact that the behaviors of the robot were coded and hard to keep consistent. After finally having words that looked perfect, the servos would sometimes change and not give constant results. Thus, sometimes the words would be straight, and sometimes they would be crooked and inconsistent. Another factor that

added into the different appearances of the robot was the fact that I had to use a plastic cover to cover up the floor, so that the marker would not mark up the tiles. This plastic cover caused slippage and also inconsistent results.

I believe that with more time and chances to change to actual stepper motors, the performances and results of my robot could be better then exhibited. The surface of the ground could also be improved by finding something that the wheels would not slip on. I think that if a thicker marker would be possible, then the resulting “Ohio” and “Gators” could be seen better also. Overall though, I feel that the robot was very successful and impressive.

Figures 22 – Script Ohio

Figure 23 – Gators

Experimental Layout and Results

The coding and experimenting of Brutus was divided up into several parts, which is evident in the program code. One of the first items that I tried to achieve was to have "Script Ohio" and "Gators" written to the point where they could be successfully determined (Figure 24). This took a great deal of time and careful testing, but finally correct results were found.

The solenoid was also a separate test because of the fact that I had to build the optoisolator circuit and run multiple tests on it. The solenoid never seemed to work correctly because of the fact that it was very small, and not much in terms of power output. It was however able to push the marker down far enough to draw the lines of the different words.

Another separate test was for the sound output of the voice playback chip. This chip became a huge pain because I had to test its functionality and understand how to use it. The initial tests and experiments were just on the breadboard, trying to record tests sounds and having it play back recordings. After finally having this successful, I had a huge problem understanding how to advance messages, and not only play the first recording each time. Finally, I was ready to place the circuit on a protoboard, which had problems in itself by the fact that the wire wrap produced messy circuits. Finally the tests with the microprocessor board were also successful and ready to use correctly.

Testing and experimenting with the CMUcam was also another huge headache because of the fact that the included documentation was not very well written and did not give accurate information on how to implement the camera. After many tests and talking with fellow students, the answer for the camera was discovered. The camera then had

problems with the lighting, background, and connecting to the microprocessor board. These problems were slowly solved with countless experiments and testing.

One of the last tests that was implemented was finding the mascots, by using the Sharp IR Distance Sensor. This sensor was probably not the ideal sensor to use for this purpose, but the only sensor that I had available and ready to use. I tested many times for the different values, and trying to position the robot it place to capture the correct image by the CMUcam. After finally positioning itself correctly, the images were accurate and determined correctly.

Implementing the tests to run smoothly together was a huge task, but taken care of accurately by the program code. Each piece of the code was written separately, in order to achieve less confusion and easier way to debug. The main code is contained in "OSUF.c", which makes the correct function calls to the many other functions. By creating all of these separate functions, testing the integrating the code together resulted in the desired outputs and tests.

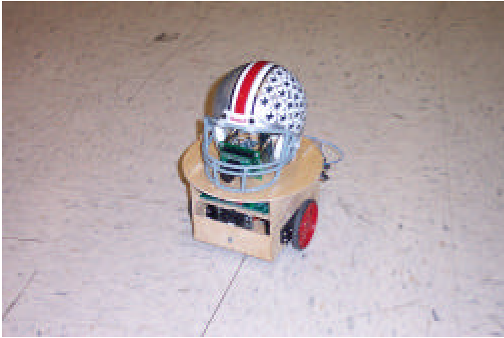
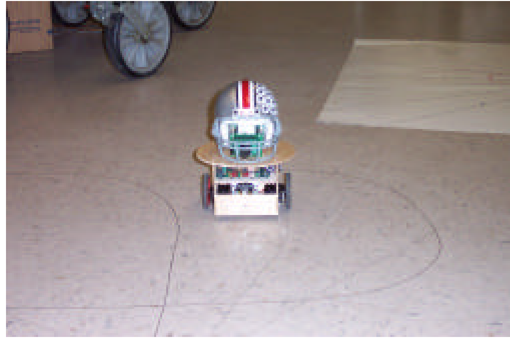

Figure 24 – Testing

Conclusion

Brutus was able to achieve every goal that I specified for it to do. Brutus was able to find and detect a mascot, being either Brutus Buckeye or Albert Alligator, use the CMUcam to determine which mascot it is, and perform "Script Ohio" and "Gators" while playing the fight song.

I feel that I was limited to the amount of other behaviors that I could implement into the robot because of the fact that I spent most of my time understanding how to use the different hardware that I had. Much time was spent testing out the different pieces of sensors, servos, microcontroller, camera, and solenoids, instead of implanting different things to improve the robot. Using the Atmel microprocessor board seemed to be a good choice because it was user friendly and had test codes and examples that could be used to understand the programs.

By finally being able to correctly implement a project together with so many different aspects of classes learned, proved to be a very valuable experience. Instead of just learning about theory and aspects to Electrical Engineering, this class allowed us to have our own thoughts and try different things. Much can be learned by doing things hands on and learning from failures and mistakes. This is also applicable to life situations and understanding how to handle different problems.

Overall, I found this robot project to be a very difficult and time consuming project, but also a very great experience. I suggest this class for everyone who actually wants to learn and enjoys doing things hands-on.

Predictions

Now is the time we discuss the real future – January 3rd, 2003 in Tempe, Arizona. The two undefeated college football teams in the country – The Ohio State University Buckeyes and the University of Miami Hurricanes will be stepping onto the field at 8pm to play for the National Championship. As this robot project is coming down to a close, it has been programmed to choose between the two best mascots and college football programs in the country – the OSU Buckeyes and the UF Gators, and while only one of these teams has the opportunity to play on January 3rd, I strongly believe that the robot knows who to choose. In the perfect world and situation, both of these teams would be playing, but for now, we'll just have to choose Brutus Buckeye.

GO BUCKEYES!!!

Documentation

The following were purchased from Mark III Electronics

(http://www.junun.org/MarkIII/Store.jsp):

- Sharp GP2D12 Distance Measuring Sensors
- Balsa Products BP148N 2BB: Standard-torque Ball-bearing Servo Motors
- Injection Molded Wheels

The following were purchased from Jameco Electronics (www.jameco.com):

- ISD 25120 Voice Record/Playback Device
- Solenoid – 12VDC 36 ohm
- Switch- D2F Series Subminiature Snap-Action Switches
- Speaker – 8 ohm

The following were purchased from Progressive Resources LLC (www.prllc.com):

- ***MegaAVR-Dev*** development board

The following were purchased from Seattle Robotics

(www.seattlerobotics.com/cmuinfo.htm):

- CMUcam

Appendices

Program Code – Ordered as follow:

OSUF.c – main code
ScriptOhio.h
ScriptOhio.c
ScriptGators.h
ScriptGators.c
wait.h
wait.c
motor.h
motor.c
ADC.h
ADC.c
LED.h
LED.c
Marker.h
Marker.c
Music.h
Music.c
Camera.h
Camera.c
FindMascot.h
FindMascot.c

OSUF.c – main code

```
#include <io.h>
#include <math.h>
#include <interrupt.h>
#include <sig-avr.h>
#include <progmem.h>

#include "motor.h"
#include "wait.h"
#include <stdlib.h>
#include <string.h>

#include "ScriptOhio.h"
#include "ScriptGators.h"
#include "ADC.h"
#include "LED.h"
#include "UART.h"
#include "marker.h"
#include "Camera.h"
#include "Music.h"
#include "FindMascot.h"

int main(void)
{
	Music_init();
	MOTOR_init();
	ADC_init();
	UART_init();
	Camera_init();

	outp(0xff,DDRC);

//	ScriptOhio();



	unsigned char IR_value;
	unsigned char BumpSensor;
	unsigned char BumpSensor2;
	unsigned char rdata;

	IR_value = ADC_getreading(0);
	BumpSensor = ADC_getreading(2);
	BumpSensor2 = ADC_getreading(3);


	while (1)
	{
		BumpSensor = ADC_getreading(2);

		while (BumpSensor > 50)
		{
			BumpSensor = ADC_getreading(2);
		}
```

```
		FindMas();


		//blink();

		rdata = ParseString();


		if (rdata < 223)
		{
			Play_UF();
			wait(1000);
			ScriptGators();
			Play_UF();
		}
		else
		{
			Play_OSU();
			wait(1000);
			ScriptOhio();
			Play_OSU();
		}
		cbi(PORTC,PC1);

	}
}
```

ScriptOhio.h

```
#ifndef SCRIPTOHIO_H
#define SCRIPTOHIO_H

extern void ScriptOhio(void);

#endif
```

ScriptOhio.c

```
#include <sig-avr.h>
#include <progmem.h>

#include "ScriptOhio.h"
#include "marker.h"
#include "music.h"

void ScriptOhio(void)
{
	MOTOR_init();


	//turn
	motor_run(25, -963,	800);  //turn
	motor_run(0, 0,		100);

	pen_down();
	wait(1000);


// "O"


	motor_run(54, -969,	1000);
	motor_run(54, -970,	1000);
	motor_run(54, -969,	1000);

	motor_run(54, -985,	700);
	motor_run(53, -985,	700);
	motor_run(52, -985,	700);
	motor_run(53, -985,	550);

	motor_run(54, -970,	500);

	motor_run(53, -985,	550);
	motor_run(52, -985,	700);
	motor_run(53, -985,	700);
	motor_run(54, -985,	700);

	motor_run(54, -969,	1000);
	motor_run(54, -970,	1000);
```

```
		motor_run(54, -969,		1000);

		motor_run(54, -985,		700);
		motor_run(53, -985,		700);
		motor_run(52, -985,		700);
		motor_run(53, -985,		550);

		motor_run(54, -970,		500);

		motor_run(53, -985,		550);
		motor_run(52, -985,		700);
		motor_run(53, -985,		700);
		motor_run(54, -985,		700);

//		motor_run(54, -985,		700);
//		motor_run(53, -985,		200);
		motor_run(52, -985,		1500);
		motor_run(53, -985,		300);
		motor_run(54, -985,		300);

//"O" finish
		motor_run(54, -969,		1000);
		motor_run(54, -970,		1000);
		motor_run(54, -969,		1000);

		motor_run(54, -970,		500);


//"h" start
		motor_run(54, -985,		300);
		motor_run(53, -985,		300);
		motor_run(52, -985,		1500);

		motor_run(54, -975,		1000);

		motor_run(52, -985,		1500);
		motor_run(53, -985,		300);
		motor_run(54, -985,		300);

		motor_run(0, 0, 100);
		pen_up();
		motor_run(0, 0, 100);
		motor_run(25, -963,		700);	//turn

		motor_run(54, -970,		600);
		pen_down();
		motor_run(54, -969,		1000);
		motor_run(54, -970,		1000);
		motor_run(54, -969,		1500);
		motor_run(54, -970,		500);
		motor_run(54, -969,		1500);
		motor_run(54, -970,		500);
		motor_run(54, -969,		1000);
```

```
		pen_up();
		motor_run(25, -972,		1100);	//spin

		motor_run(54, -969,		2000);

		pen_down();
		motor_run(70, -969,		550);
		motor_run(70, -968,		700);
		motor_run(70, -966,		3000);

		motor_run(70, -968,		700);
		motor_run(71, -969,		100);

//end "h"
		Play_OSU();

//begn "i "

		motor_run(69, -985,		100);
		motor_run(55, -985,		300);
		motor_run(52, -985,		3600);
		motor_run(53, -985,		300);
		motor_run(54, -985,		300);

		motor_run(54, -969,		2000);

		pen_up();
		motor_run(25, -972,		1100);	//spin


		motor_run(54, -969,		2000);

		pen_down();
		motor_run(60, -985,		300);
		motor_run(55, -985,		300);
		motor_run(52, -985,		3600);
		motor_run(53, -985,		300);
		motor_run(53, -985,		300);

//end " "
		motor_run(54, -970,		1000);

//"o"

		motor_run(70, -970,		300);
		motor_run(70, -969,		600);
		motor_run(70, -966,		3400);

		motor_run(70, -969,		600);
		motor_run(70, -970,		300);


		motor_run(54, -969,		1000);


		motor_run(70, -970,		300);
```

```
        motor_run(70, -969,     600);
        motor_run(70, -966,     3400);

        motor_run(70, -969,     600);
        motor_run(71, -970,     300);

        motor_run(54, -969,     500);


        motor_run(0,0, 500);

        pen_up();

//dot "i"

        Play_OSU();

        motor_run(25, -972,     150);
        motor_run(54, -969,     4000);
        pen_down();
        motor_run(25, -972,     5000);

        pen_up();
        motor_run(0, 0, 100);

}
```

ScriptGators.h

```
#ifndef SCRIPTGATORS_H
#define SCRIPTGATORS_H

extern void ScriptGators(void);

#endif
```

ScriptGators.c

```
#include <sig-avr.h>
#include <progmem.h>

#include "ScriptGators.h"
#include "marker.h"
#include "music.h"

void ScriptGators(void)
{
	MOTOR_init();

	motor_run(38, -954, 2000);

	pen_down();
	wait(1000);

///*
//"G"
	motor_run(69, -985,		100);
	motor_run(55, -985,		100);
	motor_run(53, -985,		7000);
	motor_run(53, -985,		500);
	motor_run(54, -985,		300);

	motor_run(54, -969,		300);

	motor_run(0, 0, 100);
	pen_up();
	motor_run(25, -963,		1000);  //turn

	motor_run(54, -970,		600);
	pen_down();
	motor_run(54, -969,		1000);

	motor_run(0,0,100);
	pen_up();
	motor_run(38, -954, 3000);  //reverse

	motor_run(38, -954, 3500);
	pen_down();
	motor_run(54, -969, 1500);
	pen_up();
```

```
		motor_run(54, -969, 2000);
//end "G"
//"a"
		pen_down();
		motor_run(69, -985,		300);
		motor_run(55, -985,		300);
		motor_run(51, -985,		4500);
		motor_run(53, -985,		300);
		motor_run(54, -985,		300);

		motor_run(54, -969,		700);

		pen_up();
		motor_run(25, -972,		1050);	//spin


		motor_run(54, -969,		1000);

		pen_down();



//"t"
		motor_run(69, -985,		100);
		motor_run(55, -985,		300);
		motor_run(50, -985,		2300);
		motor_run(53, -985,		200);
		motor_run(54, -985,		100);

		motor_run(54, -969,		2500);

		pen_up();
		motor_run(25, -972,		1050);	//spin

		motor_run(54, -969,		2500);

		Play_UF();
		pen_down();

//*/
		motor_run(69, -985,		100);
		motor_run(55, -985,		300);
		motor_run(50, -985,		2300);
		motor_run(53, -985,		200);
		motor_run(54, -985,		100);

//end "t"

//"o"
		motor_run(69, -969,		100);
		motor_run(69, -965,		7000);
		motor_run(69, -969,		200);


		pen_up();

//end "o"
```

```
//"r"
        motor_run(70, -950,         400);
        motor_run(54, -969,         2500);
        motor_run(0,0,    100);
        pen_down();
        motor_run(56, -969,         1500);

        pen_up();
        motor_run(0, 0, 100);
        motor_run(48, -943,         1000);  //turn

        motor_run(54, -970,         600);
        pen_down();
        motor_run(0,0,100);

        motor_run(53, -971,         2500);

//"s"
        //motor_run(0, 0, 100);
        pen_up();
        motor_run(0,0, 100);
        motor_run(25, -963,         1200);  //turn

        motor_run(54, -969,         1500);
        pen_down();
        motor_run(0,0,100);
        motor_run(54, -968,         2000);


        pen_up();
        motor_run(0, 0, 100);
        motor_run(48, -943,         1100);  //turn

        motor_run(54, -970,         1000);
        pen_down();

        motor_run(54, -969,         500);
        motor_run(60, -965,         300);
        motor_run(68, -965,         3000);


        motor_run(0, 0, 1000);
        pen_up();
}
```

wait.h

```
#ifndef WAIT_H
#define WAIT_H

extern void wait(int delay);

#endif
```

wait.c

```
#include <io.h>
 #include "wait.h"


void wait(int delay)
{
   while (delay > 0)
   {
     delay--;
     int i;
     for(i=1597;i;i--)asm("nop");
   }//while

}//wait
```

motor.h

```
#ifndef MOTOR_H
#define MOTOR_H


#define SERVO_STOP 0 //102
#define SERVO_RIGHT 1
#define SERVO_LEFT 1000
#define FULL_RIGHT 100
#define FULL_LEFT -100
#define LEFT_MOTOR 0
#define RIGHT_MOTOR 1
#define K 10

extern void MOTOR_init(void);
extern void motor_run(int leftmotor, int rightmotor, int waittime);
#endif
```

motor.c

```
#include <io.h>
#include <math.h>
#include "motor.h"
#include "wait.h"

void MOTOR_init(void)
{
        __outw(SERVO_STOP, OCR1AL);        //5 -right
        // set PWM to 10% duty cycle on channel A
        __outw(SERVO_STOP, OCR1BL);        //4 -left
        // set PWM to 10% duty cycle on channel B
        outp((1<<COM1A1) | (1<<COM1B1) | (1<<PWM10) | (1<<PWM11), TCCR1A);
        // set output compare OC1A/OC1B clear on compare match,
        outp((1<<CS11) | (1<<CS10), TCCR1B);                          // store
prescaler Clk(I/O)/8.
        sbi(DDRD, PD4); //left
        sbi(DDRD, PD5); //right

}

void motor_run(int leftmotor, int rightmotor, int waittime)
{
        __outw(leftmotor, OCR1AL);
        __outw(rightmotor, OCR1BL);

        wait(waittime);
}
```

ADC.h

```
#ifndef ADC_H
#define ADC_H


extern void ADC_init(void);
extern unsigned char ADC_getreading(int channel);

#endif
```

ADC.c

```
#include "ADC.h"

#include <interrupt.h>
#include <sig-avr.h>
#include <progmem.h>


 //Initialize the A/D converter
 void ADC_init(void)
 {
		outp((1<<ADEN) | (1<<ADPS2) | (ADPS1), ADCSR); //Initialize to use 8bit resolution for all
channels
 }//ADC_init


 unsigned char ADC_getreading(int channel) //, int refvolt)
 {
		unsigned char IR_value;

		//unsigned char temp_valueH;

		//if(refvolt == 1){
		//		outp((1<<REFS0)|(1<<REFS1)|(1<<ADLAR), ADMUX);		//use 4.95V as reference
voltage
		//}
		//else{
				outp((1<<REFS0)|(1<<ADLAR), ADMUX);			//use 2.56V as reference voltage
		//}

		if(channel == 0){}
		else if(channel == 1){
				sbi(ADMUX, MUX0);
		}
		else if(channel == 2){
				sbi(ADMUX, MUX1);
		}
		else if(channel == 3){
				sbi(ADMUX, MUX0);
				sbi(ADMUX, MUX1);
```

```
	}
	else if(channel == 4){
		sbi(ADMUX, MUX2);
	}
	else if(channel == 5){
		sbi(ADMUX, MUX2);
		sbi(ADMUX, MUX0);
	}
	else if (channel == 6){
		sbi(ADMUX, MUX2);
		sbi(ADMUX, MUX1);
	}
	else if (channel == 7){
		sbi(ADMUX, MUX2);
		sbi(ADMUX, MUX1);
		sbi(ADMUX, MUX0);
	}

	sbi(ADCSR, ADSC);

	loop_until_bit_is_set(ADCSR, ADIF);			//wait till conversion is complete
	//temp_valueH = inp(ADCH);
	IR_value = inp(ADCH);

	sbi(ADCSR, ADIF);

	return IR_value;
}//ADC_getreading
```

LED.h

```
#ifndef LED_H
#define LED_H

extern void LED(int IR_value);

#endif
```

LED.c

```
#include "LED.h"

#include <interrupt.h>
#include <sig-avr.h>
#include <progmem.h>

void LED(int IR_value)
{

	if (IR_value < 15) {
		sbi(PORTC,PC0);
		cbi(PORTC,PC1);
		cbi(PORTC,PC2);
		cbi(PORTC,PC3);
		cbi(PORTC,PC4);
		cbi(PORTC,PC5);
		cbi(PORTC,PC6);
		cbi(PORTC,PC7);
	}
	else if (IR_value < 30) {
		cbi(PORTC,PC0);
		sbi(PORTC,PC1);
		cbi(PORTC,PC2);
		cbi(PORTC,PC3);
		cbi(PORTC,PC4);
		cbi(PORT C,PC5);
		cbi(PORTC,PC6);
		cbi(PORTC,PC7);
	}
	else if (IR_value < 45) {
		cbi(PORTC,PC0);
		cbi(PORTC,PC1);
		sbi(PORTC,PC2);
		cbi(PORTC,PC3);
		cbi(PORTC,PC4);
		cbi(PORTC,PC5);
		cbi(PORTC,PC6);
		cbi(PORTC,PC7);
	}
	else if (IR_value < 60) {
		cbi(PORTC,PC0);
```

```
			cbi(PORTC,PC1);
			cbi(PORTC,PC2);
			sbi(PORTC,PC3);
			cbi(PORTC,PC4);
			cbi(PORTC,PC5);
			cbi(PORTC,PC6);
			cbi(PORTC,PC7);
		}
		else if (IR_value < 75) {
			cbi(PORTC,PC0);
			cbi(PORTC,PC1);
			cbi(PORTC,PC2);
			cbi(PORTC,PC3);
			sbi(PORTC,PC4);
			cbi(PORTC,PC5);
			cbi(PORTC,PC6);
			cbi(PORTC,PC7);
		}
		else if (IR_value < 90) {
			cbi(PORTC,PC0);
			cbi(PORTC,PC1);
			cbi(PORTC,PC2);
			cbi(PORTC,PC3);
			cbi(PORTC,PC4);
			sbi(PORTC,PC5);
			cbi(PORTC,PC6);
			cbi(PORTC,PC7);
		}
		else if (IR_value < 105) {
			cbi(PORTC,PC0);
			cbi(PORTC,PC1);
			cbi(PORTC,PC2);
			cbi(PORTC,PC3);
			cbi(PORTC,PC4);
			cbi(PORTC,PC5);
			sbi(PORTC,PC6);
			cbi(PORTC,PC7);
		}
		else {
			cbi(PORTC,PC0);
			cbi(PORTC,PC1);
			cbi(PORTC,PC2);
			cbi(PORTC,PC3);
			cbi(PORTC,PC4);
			cbi(PORTC,PC5);
			cbi(PORTC,PC6);
			sbi(PORTC,PC7);
		}
}
```

Marker.h

```
#ifndef MARKER_H
#define MARKER_H

extern void pen_down(void);
extern void pen_up(void);

#endif
```

Marker.c

```
#include <io.h>
#include <math.h>
#include "marker.h"

void pen_down(void)
{
        sbi(PORTC,PC1);
}

void pen_up(void)
{
        cbi(PORTC,PC1);
}
```

Music.h

```
#ifndef MUSIC_H
#define MUSIC_H

extern void Music_init(void);
extern void Play_OSU(void);
extern void Play_UF(void);

#endif
```

Music.c

```
#include "music.h"
#include "wait.h"

#include <interrupt.h>
#include <sig-avr.h>
#include <progmem.h>



void Music_init(void)
{
	outp(0xff,DDRB);

//stop
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	sbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

}


void Play_OSU(void)
{
	outp(0xff,DDRB);
```

```
//stop
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	sbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//reset
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//start
	cbi(PORTB,PB0);		//mode0
	cbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//play
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

/*
loop_until_bit_is_clear(PORTA, 4);

wait(100);
//repeat
//stop
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	sbi(PORTB,PB2);		//stop/reset
```

```
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//reset
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//start
	cbi(PORTB,PB0);		//mode0
	cbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//play
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

*/
}



void Play_UF(void)
{
	outp(0xff,DDRB);

//stop
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	sbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
```

```
		cbi(PORTB,PB4);
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

		wait(50);

//reset
		cbi(PORTB,PB0);		//mode0
		sbi(PORTB,PB1);		//start/pause
		cbi(PORTB,PB2);		//stop/reset
		sbi(PORTB,PB3); //playbac/record
		cbi(PORTB,PB4);
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

		wait(50);

//set mode0
		sbi(PORTB,PB0);		//mode0
		sbi(PORTB,PB1);		//start/pause
		cbi(PORTB,PB2);		//stop/reset
		sbi(PORTB,PB3); //playbac/record
		cbi(PORTB,PB4);
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

		wait(50);

//advance message
		sbi(PORTB,PB0);		//mode0
		cbi(PORTB,PB1);		//start/pause
		cbi(PORTB,PB2);		//stop/reset
		sbi(PORTB,PB3); //playbac/record
		cbi(PORTB,PB4);
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

		wait(50);

//end advance
		sbi(PORTB,PB0);		//mode0
		sbi(PORTB,PB1);		//start/pause
		cbi(PORTB,PB2);		//stop/reset
		sbi(PORTB,PB3); //playbac/record
		cbi(PORTB,PB4);
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

		wait(50);

//reset
```

```
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);
//start
	cbi(PORTB,PB0);		//mode0
	cbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//play
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

/*
//repeat

//stop
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	sbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//reset
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);
```

```
	wait(50);

//set mode0
	sbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//advance message
	sbi(PORTB,PB0);		//mode0
	cbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//end advance
	sbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);

//reset
	cbi(PORTB,PB0);		//mode0
	sbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
	cbi(PORTB,PB5);
	cbi(PORTB,PB6);
	cbi(PORTB,PB7);

	wait(50);
//start
	cbi(PORTB,PB0);		//mode0
	cbi(PORTB,PB1);		//start/pause
	cbi(PORTB,PB2);		//stop/reset
	sbi(PORTB,PB3); //playbac/record
	cbi(PORTB,PB4);
```

```
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

		wait(50);

//play
		cbi(PORTB,PB0);		//mode0
		sbi(PORTB,PB1);		//start/pause
		cbi(PORTB,PB2);		//stop/reset
		sbi(PORTB,PB3); //playbac/record
		cbi(PORTB,PB4);
		cbi(PORTB,PB5);
		cbi(PORTB,PB6);
		cbi(PORTB,PB7);

*/

}
```

Camera.h

```
#ifndef CAMERA_H
#define CAMERA_H


extern void blink(void);
extern void Camera_init(void);

extern unsigned char ParseString(void);

#endif
```

Camera.c

```
#include "Camera.h"
#include "wait.h"
#include "UART.h"

#include <interrupt.h>
#include <sig-avr.h>
#include <progmem.h>




/* blinks the CMUcam tracking light */

void blink()
{
//		PRINT("RS\r\n");
//		wait(1000);

		PRINT("L1 1\r\n");
//		EOL();
		wait(1000);
		PRINT("L1 0\r\n");
		wait(1000);


}



/*
 * reset camera
 * enable poll mode
 * enable middle mass tracking mode
 * set auto-gain and exposure of 40
 * set color tracking parameters for orange ball
 * set auto mode for tracking light
 * set raw IO mode and ACK/NCK suppression
 */
void Camera_init(void)
```

```
{
  PRINT("RS"); //tx_command("RS");
  EOL();
  blink();
  PRINT("PM 1"); //tx_command("PM 1");
  EOL();
  blink();
//  PRINT("MM 1"); //tx_command("MM 1");
//  EOL();
  blink();
  PRINT("CR 19 32 16 140"); //tx_command("CR 19 32 16 40");
  EOL();
  wait(1000); //aCore_Sleep(2000);
}



unsigned char ParseString(void)
{
		unsigned char colon;
		unsigned char numdata;
		unsigned char Sdata;
		unsigned char rdata;
		unsigned char gdata;
		unsigned char bdata;

		PRINT("RM 3\r");
		wait(100);

		PRINT("GM \r\n");

		colon = UART_ReceiveByte();
		numdata = UART_ReceiveByte();
		Sdata = UART_ReceiveByte();
		rdata = UART_ReceiveByte();
		gdata = UART_ReceiveByte();
		bdata = UART_ReceiveByte();
//		wait(100);

		blink();
		return(rdata);
}
```

FindMascot.h

```
#ifndef FINDMASCOT_H
#define FINDMASCOT_H

#define Dist 45

extern unsigned char AverageIR(void);
extern void FindMas(void);

#endif
```

FindMascot.c

```
#include <io.h>
#include <math.h>
#include <interrupt.h>
#include <sig-avr.h>
#include <progmem.h>

#include "motor.h"
#include "wait.h"
#include <stdlib.h>
#include <string.h>

#include "FindMascot.h"
#include "ADC.h"
#include "LED.h"
#include "UART.h"
#include "motor.h"

unsigned char AverageIR(void)
{
        unsigned char a1, a2, a3, a4, a5, a6;
        unsigned char average;

        a1 = ADC_getreading(0);
        a2 = ADC_getreading(0);
        a3 = ADC_getreading(0);
        a4 = ADC_getreading(0);
        a5 = ADC_getreading(0);
        a6 = ADC_getreading(0);

        if ((a3 <= a4) && (a3 <= a5) && (a3 <= a6))
        {
                a3 = 0;
        }
        else if ((a4 <= a5) && (a4 <= a6))
        {
                a4 = 0;
        }
        else if (a5 <= a6)
        {
```

```
			a5 = 0;
		}
		else
		{
			a6 = 0;
		}

		if ((a3 >= a4) && (a3 >= a5) && (a3 >= a6))
		{
			a3 = 0;
		}
		else if ((a4 >= a5) && (a4 >= a6))
		{
			a4 = 0;
		}
		else if (a5 >= a6)
		{
			a5 = 0;
		}
		else
		{
			a6 = 0;
		}

		average = ((a3 + a4 + a5 + a6) / 4);
		return(average);

}

void FindMas(void)
{
		unsigned char IR_value;

//		motor_run(54, -969,		1000);  //fast straight
//		motor_run(50, -963,	5000);  //slow straight
//		motor_run(46, -960,		5000);  //slow backwards
//		motor_run(47, -962,		5000);  //slow spin left
//		motor_run(49, -961,		5000);  //slow spin right
								//left motor stops at 48
		motor_run(0, 0, 100);

		IR_value = ADC_getreading(0);



		while (IR_value < 24)
		{
			motor_run(49, -961, 100);
			IR_value = ADC_getreading(0);

			//IR_value = ADC_getreading(0);
			//UART_Printfu16(IR_value);
			//EOL();

		}
```

```
	motor_run(49, -961, 200);

	IR_value = ADC_getreading(0);


	while (IR_value != 45)
	{
		//if (IR_value < 24)
		//{
		//	motor_run(46, -962,	100);
		//}
		//else
		if (IR_value < 45)
		{
			motor_run(50, -964,	100);
		}
		else
		{
			motor_run(46, -960,	100);
		}
		IR_value = ADC_getreading(0);

		//IR_value = ADC_getreading(0);
		//UART_Printfu16(IR_value);
		//EOL();


	}

		IR_value = ADC_getreading(0);
		//UART_Printfu16(IR_value);
		//EOL();
		wait(100);

	motor_run(0, 0, 100);


}
```

CHAMPIONS BLEED
SCARLET & GRAY
SBC
DSW
SHOE

Kroger
STATE FARM
DOMINION