The Dream Green

Final Report

<u>Project Number:</u>
May06-03

<u>Client:</u>
Charles Juel

<u>Faculty Advisors:</u>
Lamont and Patterson

<u>Team Members:</u>
Ryan Emerson - CPRE, David Goldberg - EE,
David Moline - EE, Mevan Vijthakumara - ME,
Brian Wicks - CPRE

<u>Date Submitted:</u>
3/31/2006

Table of Contents

Table of Figures

Table of Tables

Definitions

SPI (serial peripheral interface) – A serial connection between electron devices, usually a processor and peripheral. Some manufactures refer to it as Microwire.

CAN (controller area network) - A serial communication protocol originally developed by Bosch GmbH for use in vehicle networks.

LCD – liquid crystal display

USART - Universal Synchronous Asynchronous Receiver Transmitter

PIC – Programmable Integrated Circuit

1 Executive Summary

The purpose of this project was to finalize the automation of the Dream Green modifiable putting surface. The original Dream Green (the version currently for sale) is operated by manually sliding these wedges to raise/lower the surface. The break of the putting surface is changed by pushing and pulling wedges by motors, implemented by last year's senior design team. These motors move the wedges under a number of bars that support the surface. See Figure 1 for basic operating procedures on how to manually raise the surface of the original Dream Green system:

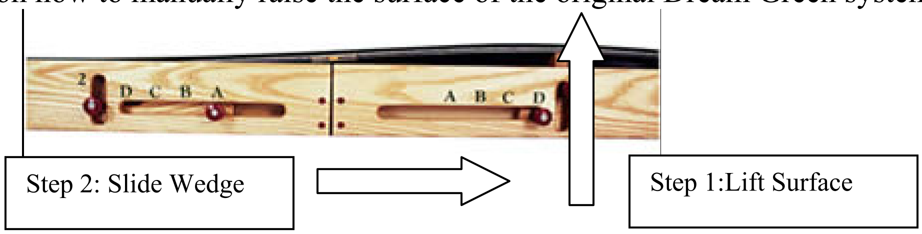


Figure 1: Manual Operation

Photo courtesy www.dreamgreen.com

The ends of the surface may also be raised and lowered by the same wedge system to add an additional putting challenge. The goal of this project was to complete the current control systems necessary to automate these adjustments. Figure 2 shows a complete view of the Dream Green apparatus:

Figure 2: Dream Green Unit

Photo courtesy www.dreamgreen.com

The design process for this team consisted of making minor modifications to the work of previous teams, as well as adding necessary components. Such modifications included switching to a motor designed for robotics due to its greater torque handling capabilities.

Added features include implementation of rotary encoders on all lifts, allowing the microcontroller to know the height each wedge. The overall goal of the team was to provide a working prototype to Mr. Juel, the client.

2 Acknowledgement

Since there have been several groups who have contributed to the Dream Green, all documentation and design work is appreciated. The current group would like to thank the previous design teams for their work. In addition, gratitude is extended to Mr. Chuck Juel for lending the Dream Green for the team's use. All of his time and effort in with past groups, as well as current and future assistance has and will prove an invaluable asset to this project.

3 Problem Statement

The Dream Green was designed to help golfers improve their putting skills by practicing on an adjustable putting surface. A series of wedges have been placed under the surface and can be adjusted to change the break of the green. Figure 3 shows the basic concept behind the motorized adjustment process:

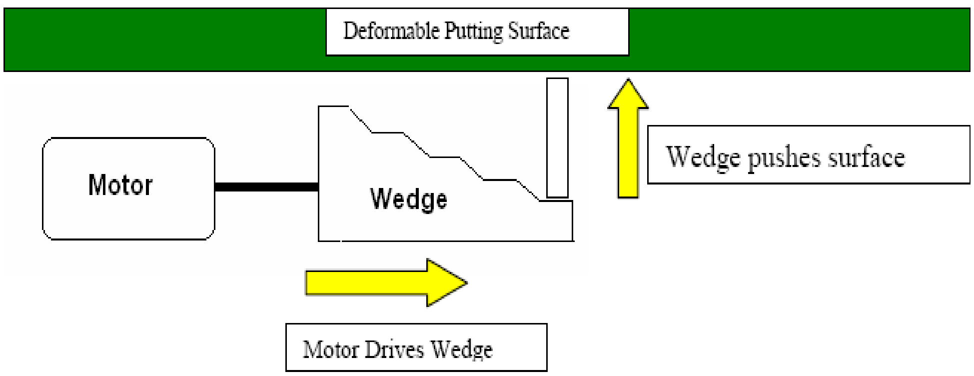


Figure 3: Motorized Operation of Wedge Adjustment System

Notice how the motor drives the "stair-step" wedge under a catch bar, effectively raising the surface. Figure 4 below shows a 3D model of the wedge used to deform the putting surface:

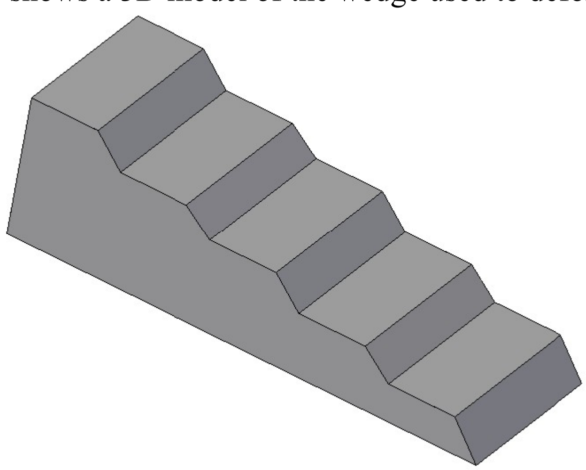


Figure 4: Wedge Detail

The client requested an automated, hands-free system to enhance the user experience and increase sells. This group's task was to complete this system.

3.1 General Problem Statement

The goal of this project was to finalize the Dream Green putting surface, building on the work of previous senior design teams. The original Dream Green required the user to manually adjust a sliding wedge system (patented by client Chuck Juel) to change the undulation of the surface (see Figure 1). The height of the ends of the green may also be adjusted to add additional variables to the apparatus.

The May05-03 senior design team faced the problem of automating these adjustments, making the Dream Green easier to use, implementing the system as shown in Figure 2. This group designed a microcontroller-based system using DC motors, making sure to keep the wedge system intact as required by the client.

The problem presented to this group was to finish the Dream Green project, and add additional features that were requested by the client. The main priority was to complete the basic automated functions of the Dream Green.

The end product must have sturdy construction suitable for heavy use, be safe for use by the general public, and fit within the space limitations of the current Dream Green. The goal was to produce a final product that is easily replicated, allowing Mr. Juel to begin marketing the automated Dream Green immediately.

3.2 General Solution Approach

To begin the Dream Green renovations, a full analysis of May05-03's materials was conducted. The current automation characteristics of the Dream Green were be tested, and notes made on several variables such as maximum adjustment time, noise, power consumption, construction characteristics, and accuracy of the wedge movement. With special attention paid to the limitations in deforming the surface, the Dream Green is unique from any other putting surface.

Because this was the final phase of the Dream Green project, the most important objective was to have a finial product ready for the client to test. The team added those features necessary to allow for reliable testing.

The ultimate team goal was to implement features that keep the Dream Green a new and exciting experience with every use. Most importantly, the team maintained the integrity of the original product, keeping fabrication and installation time and cost to a minimum.

4 Operating Environment

The operating environment for the Dream Green is indoors or other relatively controlled environments. The putting surface quickly degrades in the presence of ultraviolet light, and the Dream Green is not to be used in sunlight. The Dream Green must be able to withstand occasional spills and a reasonable amount of abuse. As a result, all electronic and mechanical components will have to be protected or located where they will not become damaged. All mechanisms must be reasonably protected from corrosion. The Dream Green must be able to withstand and operate through a reasonable amount of temperature, humidity and other environmental variations without loss of functionality. It should be feasible to use the Dream Green in an outdoor setting so long as it is not exposed to sunlight or moisture (perhaps under a tent or canopy).

5 Intended Users

The Dream Green is designed for use by a multitude of different people. The age range of the Dream Green is for all ages. The Dream Green is also designed for use by all ability levels. The weight limit of the Dream Green is 350 lbs. The Dream Green is intended for several types of game play including recreation, competition, and professional. The Dream Green has been designed to accommodate these types of play. Features have been included in the Dream Green that will increase the usability for these types of play. Recreational users desire the Dream Green to be enjoyable, easy to use and easy to operate. These recreational users also desire the Dream Green to have a minimal leaning curve. Competitive users would like consistent results in play and repeatable break settings. Professional users demand an accurate and realistic playing experience. The professional user desires the putting surface to vary in many ways in order to accommodate any putting situation. Each of the different users were considered during the design and modification of the Dream Green and its features.

6 Intended Uses

The Dream Green can be currently found in sports bars, homes, cruise ships, special events, or at college and professional sports facilities. The Dream Green is intended to be used for the purposes of putting practice, recreation and friendly tournament competition. The Dream Green is not intended to be moved while players are standing on the elevated portions of the green. The Dream Green is also not designed for people to stand on the elevated portions of the green other than the additional platforms provided with the Dream Green. The automated Dream Green will have similar uses. It can be used in sports bars, homes, cruise ships, special events or tournament, and in college and professional facilities. The automation system is simple enough to be assembled and disassembled by almost anyone. There is no need for a technical background to use the Dream Green. The Dream Green will not be used while the motors are in motion. After the motors have adjusted to their new positions, the Dream Green can be used.

7 Assumptions

Listed below are the assumptions that were taken into consideration throughout the design and redesign of the Dream Green:

1. No additional loads, including people or objects, will be on the Dream Green putting surface.
2. The Dream Green is operated on a flat, level surface.
3. All automation equipment must be installed at the factory.
4. Automation does not significantly increase the complexity of assembly and disassembly by either the end user or the builder.
5. Break adjustments are made using end lifts, wedges and other similar devices.
6. The maximum number of adjustments is the same amount as the current Dream Green.
7. The Dream Green will not be exposed to direct sunlight.
8. The Dream Green will be played in an indoor facility or tent like structure and must be hooked up to a power source.
9. All sections of the Dream Green are 4 feet long.
10. The prototype Dream Green runs off DC power.
11. The new design is not intended to be a retrofit into existing Dream Greens.

8 Limitations

Listed below are the limitations in the design and redesign of the Dream Green:

1. Wedges are used in the final design for height adjustment as necessary for patent protection.
2. The Dream Green makes full adjustment transition within 45 seconds.
3. All mechanical systems do fit within the current Dream Green enclosures, including the podium area.
4. Complexity and difficulty of the manufacturing process is minimized.
5. The Dream Green is safe in all reasonable operating conditions.
6. The automation system is reliable and requires minimal conditions.
7. The manufacturing cost of the automation system is relatively low to provide for reasonable profitability.
8. The automation system is durable enough to withstand a reasonable amount of abuse in both the shipping process and during game play.
9. The additional retail price of the automation system is under $1,000.

9 Expected End Product and Other Deliverables

Listed below are the requirements and expected capabilities of the end product delivered to the client:

1. An automation system that allows the Dream Green elevation settings to be adjusted electronically.
2. A preprogrammed course that changes holes on demand via user interface in podium unit.
3. The Design of a mechanical system used in controlling the Dream Green.
4. The ability to create a course saved in memory.
5. Instructions, drawings, and bill of materials necessary for the correct manufacturing of the Dream Green.
6. A self-contained unit.

10 Project Approach

The final phase of this project included consideration of issues from design to manufacturing.

10.1 End-Product Functional Requirements

Listed below are the requirements that were met to have a successful design:

1. The user operates in one of three modes: random wedge settings, specific wedge settings, or pre-set wedge positions.
2. Smooth, efficient wedge movement.

10.2 Design Constraints

Listed below are the guidelines for the mechanical and electrical aspects of the Dream Green:

1. All parts fit inside current Dream Green dimensions. The maximum height (with the wedges flat) is 2.5 inches.
2. The power supply is housed in the control podium.
3. Electrical and mechanical components will survive for a minimum of 5 years under normal conditions.
4. Electrical and mechanical components will survive occasional exposure to water or other liquids.
5. The entire system uses less than 15 Amps of AC current and is powered from a standard 120V circuit.
6. The time required to change positions is 45 seconds or less.
7. Additional cost are less than $1,000

10.3 Technical Approach Considerations and Results

The board-to-board communications was changed to CAN. The previous system using SPI wasn't fully implemented and was unreliable. Instead of building on SPI, it was decided to pursue and different communication standard. Many options were considered, RS-232, RS-485, and CAN. Each standard was capable of meeting the needs of this project, but CAN was the most complete protocol. CAN also included collision detection and avoidance unlike any of the other standards.

The motor position circuit was also seen to be unreliable and was replaced with a more standard solution. The past team's design used photo-diodes and LEDs to gauge how far the wedge had moved. With too much outside light, the system didn't work well. A set of tact switches was looked into as a possible solution, but it proved to be more complex to implement than originally thought. Eventually it was decided that a rotary encoder was the least expensive and easiest to employment.

The current end lift uses two motor and wedges working in tandem to lift the end. To eliminate the possibility of the motors falling out of sync, the connecting bar was removed.

10.4 Testing Approach Considerations

The first process was unit testing on the subsystems of the Dream Green. The end-lift, communications protocol, and positioning system will need to be tested individually. The second process was to integrate the subsystems to test the system as a whole. With all the system together, a working prototype was built and tested.

11 Detailed Design

The flowing is a description of the mechanical and electrical components of the automated Dream Green.

11.1 Previous Designs

This section describes the previous designs considered and/or tested in the project.

11.1.1 Side Lift Assembly

The side lift assembly originally consisted of a wedge, threaded shaft, coupler, threaded insert and motor. The driveshaft of the motor was coupled with the threaded shaft. The threaded shaft is then driven through the wedge via a threaded insert. The rotational forces were converted to linear motion. This assembly was designed in its entirety by previous design teams.

Problems that the team encountered with this setup were numerous. First and foremost, the motors used in this design were underpowered. The wedge was driven directly into the wooden crossbeam, resulting in major friction losses. In addition, the wedge was driven, unguided, into the crossbeam which resulted in lateral forces on the drive shaft and motor. It should also be noted that there were no controls in place to control the position of the wedges.

The mechanical concept of this design was correct, but it was felt that there was much to be desired in this previously used side lift assembly. Plus, all controls were still needed to automatically sense the position of each wedge.

11.1.2 End Lift Assembly

The main difference in the end lift is that the entire section must be raised and lowered, as opposed to creating a slope at a single point as it is with the side lifts. This results in a load that is far greater than for the side lifts. The initial end lift assembly was identical to the side lift in that each wedge was driven by its own motor. The main problem that the team noted in the way previous groups had with this set up is that the motors were wired in parallel. In testing, it was noticed that this parallel configuration made control nearly impossible because each motor ran at slightly different speeds. This meant that one wedge would always travel farther than the other. The group previous to the current team obviously had noted this because they attempted to add resistors in series with the faster motor in hopes of choking the current delivered to the motor. The team felt this tactic to be an ineffective and impractical solution, and redesign was required.

Because of the problems that were noted with the dual motor design, the team felt that going to a single motor would be a far more effective solution from a controls aspect. The team spent a vast amount of time devoted to research of parts and designing an end lift that would lift the heavier load with a single motor. Once a motor capable of handling the load was discovered, it was decided to go with a configuration that had the single motor drive a crossbar at its center point, with wedges positioned at the ends of the crossbar. U-shaped aluminum pieces were used to guide the wedges while the crossbar was held straight by guide shafts. The figure below shows the design:

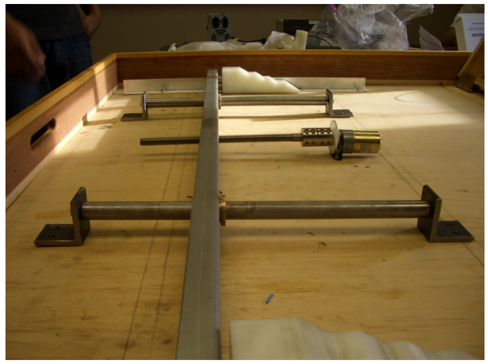


Figure 5: Previously Considered End Lift

This design, on paper, seemed as if it would work great. However, the main problem is that is was a very difficult part to manufacture. The most important component of the design was the guide shafts on the crossbar which were intended to eliminate twisting. The bearings that created the interface between the shafts and the crossbar had to be a perfect right angle to be effective. As it was discovered, that is very difficult to do in practice. The guide shafts on the crossbar were not perfectly parallel, and as a result, did not allow the motor to freely slide the apparatus forward and backward.

11.2 Final Designs

This section describes the final designs used in the final prototype.

11.2.1 Side Lift Assembly

Once motors were found with a high enough torque capability, the team was able to effectively operate the side lifts without making any significant changes to the design of the previous teams. The key was finding a motor that had enough power to get the job done.

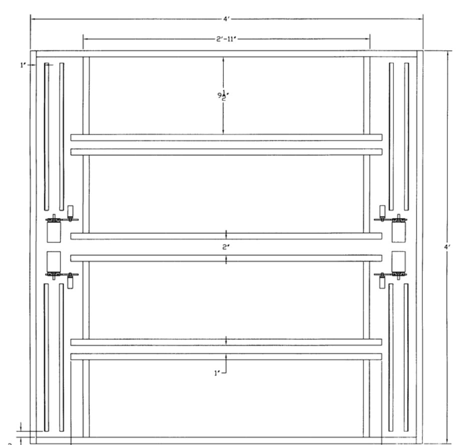


Figure 6: Side Lift Schematic

11.2.2 End Lift Assembly

After fully re-evaluating the approach to the end lift, the team decided to go back to a two-motor approach that was very similar to the design of previous teams. The main difference is that the design ran each motor independent of the other, at least in how it was wired. By connecting each motor separately to the power supply each motor was able to draw maximum current, allowing each to operate at close to maximum speed. The team just configured the controls such that each motor would always be going to the same wedge position, which allowed the load to be split. Rather than get creative with hardware, the team felt it would be easier and more cost effective to get creative on the software side to achieve the proper end result of a functional end lift. This method did prove to be effective, the team are confident that the client will be able to replicate it much easier than with the more complex crossbar design.

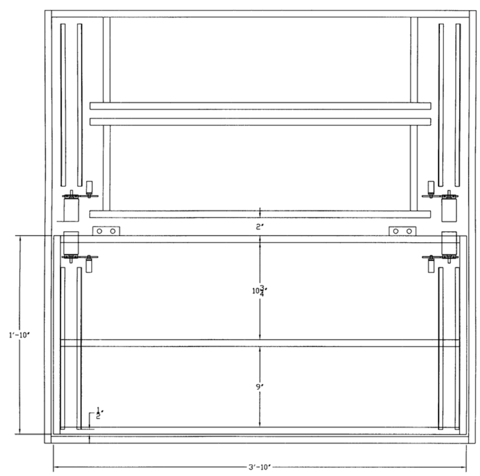


Figure 7: End Lift Schematic

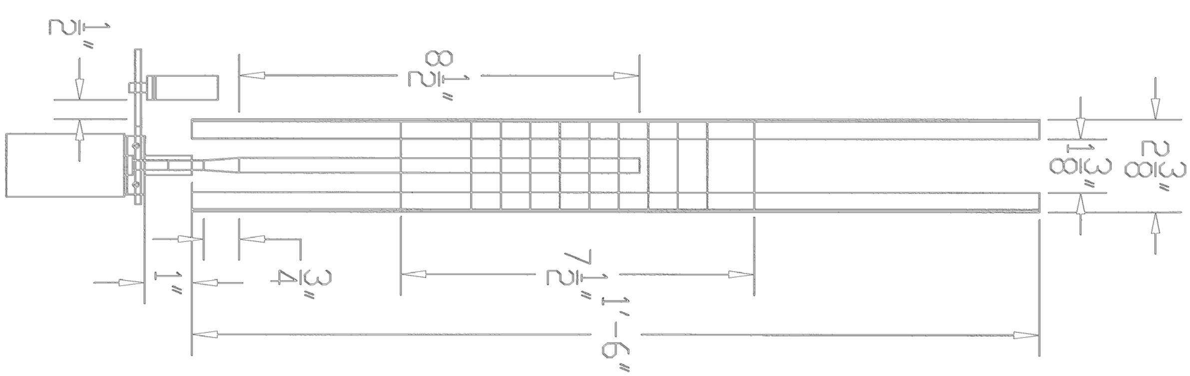


Figure 8: Detail of Wedge Positioning Apparatus

11.3 Key Components

Below is a description of the main components used in the final prototype.

11.3.1 Motor

Power thread calculations were performed to determine torque requirements. The side-lift motors must lift approximately ½” x 2 ¼” x 4’ section of wood. Given the small load requirements of the side lifts, the torque requirement for the motor is minimal. Currently, the design incorporates motors purchased by previous groups. It would be preferable and more cost effective to research smaller geared motors to replace the existing motors. The end lift is expected to lift a sheet of plywood, green support material, and the putting surface. Given the large torque necessary to lift such heavy loads, a high torque, geared motor was used. Both assemblies will require motor protection circuits to prevent burn out. However, for the prototype, four similar motors were used.

11.3.2 Wedge

Ultra-High Molecular Weight (UHMW) plastic was chosen for the wedges due to its frictional properties. Sharp edges on the wedge are to be filleted and contact surfaces will be smoothed to reduce friction. The wedge may be modified with permission from Mr. Chuck Juel. The mechanical advantage of the wedge would be increased by reducing the slope of the inclines. This would minimize the required linear force and resulting stresses on the motor. As a result, the reliability of the overall system may be increased. In addition, unnecessary and costly material can be eliminated to reduce manufacturing costs.

11.3.3 Threaded Shaft

ACME threaded rods are to be used to transfer power between the motors and the wedges. The ACME threaded rods increase power transfer capabilities and reduces the potential for binding when compared to traditional threaded rods. Square-cut threaded rods are not suitable because of their increased potential for catastrophic binding. The use of a geared motor for the end lift reduces the rotational speed of the motor considerably. Therefore, it is desirable to reduce the number of threads per inch to reduce the time requirements for adjustment.

11.3.4 Threaded Insert

The material of the insert is to be determined but will be a metal with natural lubricative properties such as red bronze or brass. The length of engagement for the threaded insert is not as critical since there are guide rails present in the design. The insert will be tapped to match the ACME threaded shaft. Initial lubrication of the threaded insert will be necessary to reduce friction between the threaded shaft and the insert. A lubricant will be chosen such that the system will require minimal, if at all, reapplication.

11.3.5 Gears

Previous groups had attempted to utilize external gears to drive the end lift wedges. Instead, geared DC motors were chosen because of their simplicity. A motor was chosen that met the size, torque, and rotational speed requirements that contained internal gearing. As such, external gears are no longer necessary for meeting torque requirements. However, in order to use rotary encoders to measure wedge position, gears were mounted to the threaded shaft. This gear was meshed with a gear mounted to a rotary encoder.

11.4 Motor Controller Circuit

Additional circuitry is required to correctly position the wedge using the motor. This is implemented by using a microcontroller, H-bridge circuit, rotary encoder and micro-switch. The schematic for this circuit is shown in Appendix B – Figure 13. To reduce manufacturing costs two motor control circuits are implemented on one printed circuit board.

11.4.1 Microcontroller

The microcontroller selected is the Microchip PIC18F248 because of it ease of use, built-in CAN controller, ADCs, timers, and digital outputs. To avoid extra costs, this model is most basic that meats all of the team's design criteria. The PIC18F248 has been used for the testing and implantation of The Dream Green, but since selection has been replaced with a new model, PIC18F280. The PIC18F2480 is pin-compatible and has all the features of the PIC18F248. However, to use the PIC18F2480 the code must be recompiled in PIC C18 to work correctly.

11.4.2 H-Bridge

The PIC will be able to enable and change directions through the H-bridge circuit. Previous teams had built their own circuit with relays. However to reduce space, complexity, and cost it has been replaced with an integrated H-bridge controller from National Semiconductor, the LMD18200. The LMD18200 is a 3A H-bridge circuit and includes short circuit protection, thermal flag, thermal shutdown, short circuit protection, and current sensor. The LMD18201 is a similar part that is pin compatible to the LMD18200, but doesn't have the current sensor.

11.4.3 Rotary Encoder

The position of the motor is monitored with the PIC using the pulses from a rotary encoder mounted on the output shaft of the motor. The rotary encoder pulses a given number of time of each revolution of the motor. Each pulse is counted using the external interrupt of the PIC. The number of revolutions to each position is known and hard coded into the PIC so it can determine where the wedge is at any given time. There is some uncertainty in using the rotary encoder because missed pluses add up over time. However the motion of the wedge doesn't have to be extremely precise because each stop is approximately an inch. One inch is translates into many revolutions of the motor. The position of the motor is also zeroed each time the device starts up to avoid long term errors.

Then encoder selected is a Panasonic ECG series encoder (EVE-GE1F2012B) with 12 pulses per revolution and a 12mm shaft with detent. This model was selected mainly because of its low cost. Any comparable rotary encoder, vertical or shaft mount, could be used. However, replacing it with an encoder that doesn't have 12 pulses per revolution will require a change to the software. Using an encoder with a higher resolution, more than 64 pulses per revolution, may be more than the PIC microcontroller can handle.

11.5 User Interface Circuit

The automation of the Dream Green allows the user to easily control the features of the Dream Green. It has an LCD display, keypad for input, and is controlled by a microcontroller. The user uses this to send commands, like moving a motor or sending the break one of pre-defined positions, to the Dream Green. The menus are designed to be user-friendly and hide much of the technical implementation from the user. The schematic for this circuit is shown in Appendix B – Figure 15.

11.5.1 Microcontroller

Because of the additional I/O requirements of the keypad and LCD, the PIC18F248 is not capable of running the user interface. The PIC18F480 was chosen because it is a member of the same family as the PIC18F248 but has additional digital I/0 to accommodate the keypad and LCD. The PIC18F448 also has an integrated USART controller that can but used to interface with a computer via RS-232.

11.5.2 Display

The information the user sees is presented on a 20x4 character LCD made by Lumex (LCM-S02004DSF). The LCD is used to prompt the user for input, if they want to play a game, one hole, random break, etc. The model chosen is a standard 20x4 character LCD and models from other manufactures should be compatible.

11.5.3 Input

The user has a 16-key keypad (Grayhill 96BB2-006-F) is what is user uses to respond to the messages displayed on the LCD. The keypad only has keys labeled 0-9 and A-D, so care has been taken simplifies all inputs. Since the pin-outs for keypads like this aren't standardized, changing models may require changes to software.

11.5.4 External

To be able to update the predefined holes from a personal computer and external connection is necessary. Since the PIC includes a USART controller, adding an RS-232 was simple and cost effective. The RS-232 single is regulated with a Maxim MAX233A transceiver.

11.5.5 Communications

To enable commutations between the user inter face circuit and the motor controller circuits a CAN (Controller Area Network) bus has been implemented. CAN was selected because of its addressing, fault tolerance, and error detection.

Each CAN packet begins with a 10-bit address, allowing 1024 unique devices on the bus. Since this the maximum number of devices need for the largest Dream Green is 32, the address segment also includes the packet type for this implementation.

Bit	9	8	7	6	5	4	3	2	1	0
Use	*Device Identifier*						*Packet Identifier*			

Device	Identifier
Master	32
Slave	0-31

Packet	Identifier	Description
Reserved	0	Sent on startup
ACK	1	Acknowledgement a packet is received.
PING	2	Sends packet back to the sender.
MOVE_WEDGE	3	Command to move a wedge to a new position.
STOP_WEDGES	4	Stops all wedges.
MOVE_COMPLETE	5	Signal that the wedge has completed its move and is ready for the next hole
POSITION_ERROR	6	The wedge cannot reach the requested position.
CURRENT_ERROR	7	The motor has drawn too much current and stopped.
TEMP_ERROR	8	The H-bridge has over-heated and stopped.
TIME_ERROR	9	The motor has been on longer than the maximum allowed time.

Each device is given an identification number from 0-32, the motor controller boards use IDs 0-31, and 32 is reserved for the motor control board. All boards only accept commands that are sent to their address, any other packets will be ignored. When a control board receives a move command directed at its address begins moving the wedge. When the motor stops it sends a reply back to the interface telling it that it is done or why it failed.

12 Implementation Process

In the implementation of the components many problems were encountered in developing a final product. This section breaks down our implementation process into three sections:

1. Mechanical implementation
2. Electrical implementation
3. Suggestions for improvement

12.1 Mechanical Implementation

There were several problems that the team ran into while implementing the mechanical components in the final design. The first problem the team encountered was handed down by the previous design team, and it had to do with a manufacturing error in the wedges. In order for the motor to drive the wedge, the back of the wedge had to be drilled out so a threaded insert could be pressed into the plastic. Ideally, the screw driving the wedge would be parallel with the ends of the wedge itself. However, this was not the case as several of the wedges ran "pigeon toed," meaning that the edges of the wedge were not in alignment with the screw. This made it very difficult to effectively guide the wedges using the aluminum guide rails.

The team could have solved this problem in two ways: one solution would be to mill the sides of the wedge to run parallel with the screw, and the other would be to mount the aluminum guide rails parallel to the screw anyway, but leaving space so the motion of the wedge would not be hindered, but might twist slightly. The team went with this option, while not ideal, it was felt that leaving some space would not be overly detrimental to performance, and may even allow some room for error.

A second major problem that the team ran into was in mounting the gears to the screws and encoder devices. The team was left with plastic gears, some of which were warped and had a very shallow catching area in the teeth of the gear. In testing of automation these gears slipped, and as a result, the encoders did not get an accurate count. This caused the controllers to think the wedge was in a position that it was not.

To solve this problem, it was necessary to replace the gears. Rather than use plastic gears that were intended for torque generation, the team switched to metal spider gears. These gears are designed to reduce the effects of twisting and slip. Once these motors were implemented, the team noticed an immediate improvement in encoder performance.

12.2 Electrical Implementation

Implementing the electronics was fairly difficult because the designs left by the previous teams were incomplete or unreliable. To correct this, many components were replaced and the software was rewritten from scratch, but the original concept of a central master control board with slave boards controlling each pair of motors.

The communications standard used by the previous teams, SPI, could have been used successfully, but it was determined that the CAN protocol would be easier to implement and would be more immune to noise from the brushless DC motor. Using CAN required adding and additional transceiver, but cost was fairly low and seemed to be justified by the noise protection, and collision detection.

The transition to CAN also required a change in microcontroller. The PIC16F877 was previous used because it was a good general purpose microcontroller, but lacked an integrated CAN controller. The PIC 18 series has several models with CAN controllers. Eventually the PIC18F248 and PIC18F448 were chosen because were the least expensive with CAN controllers and the other I/0 needed.

The last major change made was to replace the LED/photo-diode positioning system created by the previous team. Since the circuit didn't react well to changing light conditions, it was decided to replace it. Many other applications have used rotary encoders and it was decided that they would work well with this application. The encoders worked well, they have been more accurate and reduced the component count.

12.3 Suggestions for Improvement

The biggest thing that would have helped the implementation process would be to have fabricated the parts specific to the design. However, the nature of the project was to complete the work of previous teams. When the team used parts that were given by other teams, the team may or may not have been using them as intended when they were fabricated. One example is with the wedges, and the fact that the insert was not aligned parallel to the edge of the wedges. The previous team was not using rails to guide the wedge, and as a result did not have this in mind when drilling out the wedge. They probably wanted to be straight, but may not have put the same attention as the team would have knowing that guide rails would be used.

Overall, if the team would have been able to see the project through from inception, the team certainly would have had implementation issues, but likely not the same ones that were encountered.

13 End-Product Testing

In implementing the final design, several tests were conducted to confirm that the team met the design requirements laid out by the client. The following tests were conducted to evaluate the quality of the end product:

13.1 Stress Testing

In order to make sure that the motors can handle the minimal lifting of the surface structure and carpet, testing was conducted to make sure each motor could handle the load. While the team wanted to make sure not to overheat the motor, the team did want to get an idea of the actual stall current to determine a correct circuit breaker size. From testing, it was clear that the motors could handle the minimum load just fine, but if someone were to stand on the green during adjustment the motor would certainly stall. It was decided through testing that a 2 amp breaker would provide sufficient over current protection. Because the team was extra cautious during testing of blowing a motor, this testing was not designed to find out its maximum lifting capabilities. Rather, it was to design the motor protection circuit.

13.2 Game Play

This was by far the most important part of the testing. Once the team wa confident in the positioning capabilities of the controls, the Dream Green was put to the real test. The team actually used the Dream Green as it would be used by the client and consumer. Repeated use is the most effective way to work out any bugs in the system that may not be immediately apparent. From the repeated testing, it was found that it would be necessary to have the Dream Green "re-zero" itself at startup. That way, if the encoder gets off by small amounts during extended use, the positioning of the wedge would not get progressively worse. This is a prime example of bugs that can only be found when actually using the Dream Green.

End product testing is truly an ongoing process. The team is available to the client following final completion of the project for consultation purposes.

14 Project End Results

Much time and effort went in to discovering what ideas worked and what ideas did not work for the automated Dream Green. The biggest discovery was that a single motor end-lift cannot be implemented with the given resources. The single motor end-lift was not cost effective and produced many additional problems. The final decision for a dual motor end-lift was made when the single motor end-lift could not be reproduced. With the implementation of a two motor end-lift, all sections of the Dream Green are basically the same. Since all sections are very similar the automated Dream Green will be easier to reproduce. The two motor end-lifts will be much more cost effective and easier to assemble then its one motor counterpart. Along with the completed fully-functional Dream Green, there are several other documents and instructions included. These documents and additional items are as described below.

14.1 Assembly Instructions

Detailed assembly instructions accompany the Dream Green. These assembly instructions tell our client how to reproduce the automated Dream Green. This set of assembly instructions includes two main sections:

1. List of Materials
    a. Motors and wedges
    b. Threaded rods/gears/couplers
    c. Various nuts/bolts/nails/screws etc.
    d. Microcontroller parts ie. circuit boards
2. Step-by-step assembly
    a. Machining parts
    b. Attaching parts
    c. Measurements

These assembly instructions include a detailed step-by-step instruction set for the Dream Green including pictures were needed. The machining section includes all parts that need to be machined and how to machine them. The measurements section includes pictures with exact measurements on where to attach the different components of the Dream Green.

14.2 End-user Instructions

The end-user instructions are a set of instructions for the users of the Dream Green. This set of instructions includes the following:

1. Game play instructions
2. Basic assembly
3. Upkeep and maintenance
4. Safety precautions
5. Warnings
6. Troubleshooting tips

The end-user instructions were designed to provide the end customer with a complete guide on how to use and care for their automated Dream Green.

15 Resource Requirements

15.1 Personnel Effort Requirements

The following table lists the projected hours that would be spent by each of the team members. This is an estimated personnel effort budget only.

Table 1: Initial Estimated Personnel Effort Requirements

Hours per Job per Individual	Technology Evaluation	End-Product Design	End-Product Prototype Implementation	End-Product Testing	End-Product Documentation	End-Product Demonstration	Project Reporting	Total
Ryan E	30	20	70	20	20	5	15	180
David G	30	20	70	20	20	5	15	180
David M	30	20	70	20	20	5	15	180
Mevan V	30	20	70	20	20	5	15	180
Brian W	30	20	70	20	20	5	15	180
Total	150	100	350	100	100	25	75	900

Table 2: Revised Estimated Personnel Effort Requirements

Hours per Job per Individual	Technology Evaluation	End-Product Design	End-Product Prototype Implementation	End-Product Testing	End-Product Documentation	End-Product Demonstration	Project Reporting	Total
Ryan E	30	30	70	20	30	5	15	200
David G	15	25	35	20	30	5	15	145
David M	15	15	35	40	30	5	15	155
Mevan V	30	30	70	20	30	5	10	195
Brian W	15	25	35	40	30	5	30	180
Total	105	115	245	140	150	25	85	865

Table 3: Final Personnel Effort Requirements

Hours per Job per Individual	Technology Evaluation	End-Product Design	End-Product Prototype Implementation	End-Product Testing	End-Product Documentation	End-Product Demonstration	Project Reporting	Total
Ryan E	30	30	80	20	30	4	15	209
David G	28	26	75	20	30	5	15	199
David M	27	26	75	20	30	5	15	198
Mevan V	25	28	74	20	30	4	10	191
Brian W	20	20	35	20	30	4	50	179
Total	130	130	339	100	150	22	105	976

15.2 Other Resource Requirements

Listed below is a table of outside resources that the team will use. Some resources reoccurring from last year are for additional materials needed to complete the entire prototype.

Table 4: Initial Estimated Resource Requirements

Resource	Reason
Senior Design Lab	Space and equipment will be needed for constructing the Dream Green Prototype.
Dream Green	Will be used to implement design and used for testing.
Parts	Additional parts will be needed to replace burnt out parts from previous teams and new parts will be needed for motor protection, motor control, user interface, power interface, and circuit board creation.
Machine Shop	Used to make additional parts for the prototype.

Table 5: Adjusted Resource Requirements

Resource	Reason	Costs Related
Senior Design Lab	Space and equipment will be needed for constructing the Dream Green Prototype.	Part of the Class
Dream Green	Will be used to implement design and used for testing.	Donated
Parts	Additional parts will be needed to replace burnt out parts from previous teams and new parts will be needed for motor protection, motor control, user interface, power interface, and circuit board creation.	See Section 14 Other Resource Requirements
Dream Green Replacement Parts	Structural components of the Dream Green needed to be replaced due to past team's design idea trials.	$61.53
Deliverable Printing	Documents will need to be printed to be turned in.	$29.74
Machine Shop	Used to make additional parts for the prototype. See Section 13 – Manufacturing for costs.	$21.30
Total		$112.57

Table 6: Final Resource Requirements

Resource	Reason	Costs Related
Senior Design Lab	Space and equipment will be needed for constructing the Dream Green Prototype.	Part of the Class
Dream Green	Will be used to implement design and used for testing.	Donated
Parts	Additional parts will be needed to replace burnt out parts from previous teams and new parts will be needed for motor protection, motor control, user interface, power interface, and circuit board creation. - See Section 15.3, Tables 7 & 8 for details.	$3675
Deliverable Printing	Documents will need to be printed to be turned in.	$30
Total		$3705

15.3 Financial Requirements

Listed below is a table of estimated costs for the components and labor. Some costs reoccurring from last year are for additional materials needed to complete the entire prototype.

Table 7: Motor Controller Board Parts List

Motor Controller Board Parts List						
Description	Manufacturer	Part Number	Distributor	Qty	Unit Cost	Total Cost
IC LDO V REG W/DELAY TO-220-5	National Semiconductor	LM9071T	Digi-Key	1	$2.48	$2.48
IC PIC MCU FLASH 8KX16 40DIP	Microchip Technology	PIC18F248-I/P	Digi-Key	2	$5.99	$11.98
IC TRANSCEIVER CAN HI-SPD 8-DIP	Microchip Technology	MCP2551-I/P	Digi-Key	2	$1.48	$2.96
IC H BRIDGE 3A 55V TO-220	National Semiconductor	LMD18200	Digi-Key	4	$11.69	$46.76
CRYSTAL 10.000 MHZ HC49/US	Citizen America Corporation	HC49US10.000MABJ	Digi-Key	1	$0.70	$0.70
CAP .1UF 50V 20% CER RADIAL	Kemet	C315C104M5U5CA	Digi-Key	3	$0.16	$0.48
CAP CERAMIC 18PF 50V NP0 1206	Kemet	C1206C180J5GACTU	Digi-Key	6	$0.17	$1.03
RES 1.0M OHM 1/4W 5% CARBON FILM	Yageo America	CFR-25JB-1M0	Digi-Key	8	$0.28	$2.24
RES 120 OHM 1/4W 5% CARBON FILM	Yageo America	CFR-25JB-120R	Digi-Key	1	$0.28	$0.28
RES 10 OHM 1/4W 5% CARBON FILM	Yageo America	CFR-25JB-10R	Digi-Key	1	$0.28	$0.28
CONN HEADER 4POS 7.5MM R/A TIN	Molex	43160-1104	Digi-Key	1	$3.14	$3.14
CONN HEADER 5POS 7.5MM R/A TIN	Molex	43160-1105	Digi-Key	4	$3.62	$14.48
SWITCH 3 POS DIP RT ANG SLIDE	E-Switch	KAS2103ET	Digi-Key	1	$0.94	$0.94
BUZZER PIEZO ELEMENT 4.4KHZ 27MM	CUI Inc	CEB-27D44	Digi-Key	1	$1.10	$1.10
CUSTOM PRINTED CIRCUIT BOARD	Advanced Circuits	N/A	Advanced Circuits	1	$33.00	$33.00
ASSEMBLY	Screaming Circuits	N/A	Screaming Circuits	1	$40.00	$40.00
					Total	$161.85

Table 8: User Interface Board Parts List

User Interface Board Parts List						
Description	Manufacturer	Part Number	Distributor	Qty	Unit Cost	Total Cost
IC LDO V REG W/DELAY TO-220-5	National Semiconductor	LM9071T	Digi-Key	1	$2.48	$2.48
IC PIC MCU FLASH 16KX16 40DIP	Microchip Technology	PIC18F485-I/P	Digi-Key	1	$10.90	$10.90
IC TRANSCEIVER CAN HI-SPD 8-DIP	Microchip Technology	MCP2551-I/P	Digi-Key	1	$1.48	$1.48
MAX233ACPP	Maxim Integrated Products	MAX233ACPP	Digi-Key	1	$10.70	$10.70
CRYSTAL 12.000 MHZ HC49/US	Citizen America Corporation	HC49US12.000MABJ	Digi-Key	1	$0.70	$0.70
CAP .1UF 50V 20% CER RADIAL	Kemet	C315C104M5U5CA	Digi-Key	3	$0.16	$0.48
CAP CERAMIC 18PF 50V NP0 1206	Kemet	C1206C180J5GACTU	Digi-Key	4	$1.71	$6.84
RES 1.0M OHM 1/4W 5% CARBON FILM	Yageo America	CFR-25JB-1M0	Digi-Key	8	$0.28	$2.24
RES 120 OHM 1/4W 5% CARBON FILM	Yageo America	CFR-25JB-120R	Digi-Key	1	$0.28	$0.28
RES 10 OHM 1/4W 5% CARBON FILM	Yageo America	CFR-25JB-10R	Digi-Key	1	$0.28	$0.28
CONN HEADER 4POS 7.5MM R/A TIN	Molex/Waldom Electronics Corp	43160-1104	Digi-Key	1	$3.14	$3.14
CONN USB RT ANG RECPT TYPE B WHT	Molex/Waldom Electronics Corp	67068-9000	Digi-Key	1	$2.01	$2.01
LCD MOLULE 20X4 CHARACTER W/LED	Lumex Opto/Components Inc	LCM-S02004DSF	Digi-Key	1	$31.50	$31.50
KEYPAD 16 KEY FRONT PANEL MNT	Grayhill Inc	96BB2-006-F	Digi-Key	1	$12.87	$12.87
CUSTOM PRINTED CIRCUIT BOARD	Advanced Circuits	N/A	Advanced Circuits	1	$33.00	$33.00
Assembly	Screaming Circuits	N/A	Screaming Circuits	1	$52.20	$52.20
					Total	$171.10

Table 9: Initial Estimated Financial Budget

Item	Estimated Cost without Labor	Estimated Cost with Labor
Computer Chips and LCD	$100	$100
Wires	$25	$25
Gears	$75	$75
Motors	$50	$50
Bars	$25	$25
Blocks	$125	$125
Fasteners	$25	$25
Manufacturing – Student Workspace	$50	$50
Labor at $10/hr		$9000
Total Cost	**$475**	**$9475**

Table 10: Revised Financial Budget

Item	Revised Cost without Labor	Revised Cost with Labor
Circuit Boards	$954	$954
Wires	$50	$50
Gears	$75	$75
Motors	$90	$90
Wedge Material	$300	$300
Fasteners	$25	$25
Manufacturing – Student Workspace	$100	$100
Labor at $10/hr		$8650
Total Cost	**$1594**	**$10244**

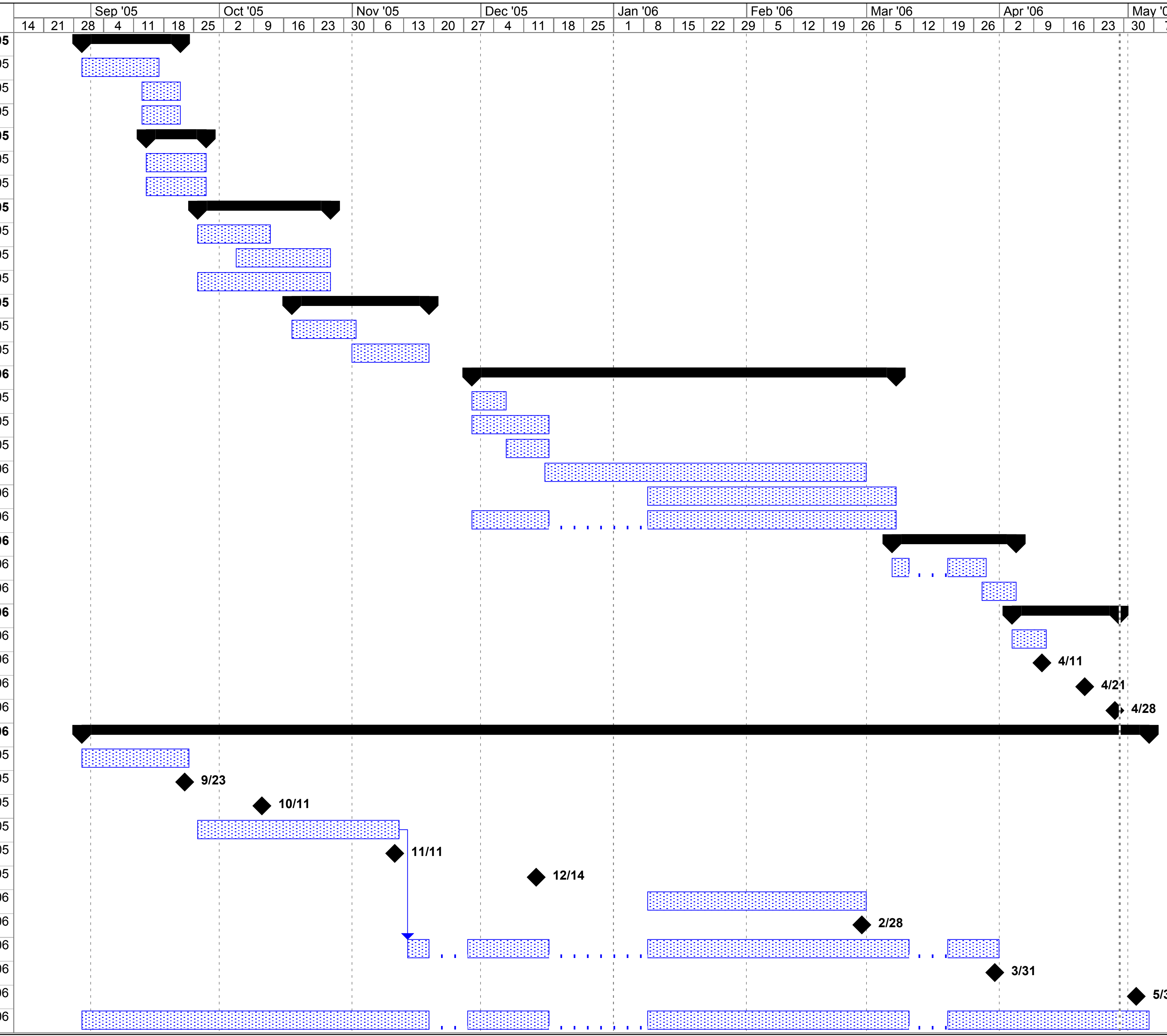
Table 11: Final Financial Budget

Item	Section Cost x5 sections	Interface Costs	Total without Labor	Total with Labor
Circuit Boards	$162	$172	$982	$982
Wires	$26	$18	$148	$148
Gears	$40		$200	$200
Motors	$120		$600	$600
Wedge Material	$54		$270	$270
Fasteners	$10		$50	$50
Other Parts	$35		$175	$100
Manufacturing	$250		$1250	$1250
Labor at $10/hr				$9760
Total Cost	**$697**	**$190**	**$3675**	**$13,435**

16 Team Gantt Charts

Below are the Gantt charts showing the original Dream Green team project timeline, the adjusted timeline, final timeline and the deliverables timeline.

ID		Task Name	Duration	Start	Finish
1		**Problem Definition**	**17 days?**	**Tue 8/30/05**	**Wed 9/21/05**
2		Problem Definition Completion	14 days?	Tue 8/30/05	Fri 9/16/05
3		Confirmation of End-User(s) and End-Use(s)	7 days?	Tue 9/13/05	Wed 9/21/05
4		Review and Identification of Project Constraints	7 days?	Tue 9/13/05	Wed 9/21/05
5		**Technology Considerations and Selection**	**10 days?**	**Wed 9/14/05**	**Tue 9/27/05**
6		Review and Modification of Technologies	10 days?	Wed 9/14/05	Tue 9/27/05
7		Technology Selection	10 days?	Wed 9/14/05	Tue 9/27/05
8		**End Project Design**	**23 days?**	**Mon 9/26/05**	**Wed 10/26/05**
9		Review of Design Requirements	13 days?	Mon 9/26/05	Wed 10/12/05
10		Design Process	16 days?	Wed 10/5/05	Wed 10/26/05
11		Documentation of Design	23 days?	Mon 9/26/05	Wed 10/26/05
12		**End-Product Prototype Implementation**	**24 days?**	**Tue 10/18/05**	**Fri 11/18/05**
13		Identification of Prototype Limitations and Substitutions	11 days?	Tue 10/18/05	Tue 11/1/05
14		Implementation of Prototype End Product	14 days?	Tue 11/1/05	Fri 11/18/05
15		**End Product Testing**	**71 days?**	**Tue 11/29/05**	**Tue 3/7/06**
16		Test Planning	6 days?	Tue 11/29/05	Tue 12/6/05
17		Test Development	14 days?	Tue 11/29/05	Fri 12/16/05
18		Test Execution	8 days?	Wed 12/7/05	Fri 12/16/05
19		Client Testing	53 days?	Fri 12/16/05	Tue 2/28/06
20		Test Evaluation	42 days?	Mon 1/9/06	Tue 3/7/06
21		Documentation of Testing	56 days?	Tue 11/29/05	Tue 3/7/06
22		**End Product Documentation**	**21 days?**	**Tue 3/7/06**	**Tue 4/4/06**
23		Development of End-User Documentation	11 days?	Tue 3/7/06	Tue 3/28/06
24		Delopment of Maintenance and Support Documentation	6 days?	Tue 3/28/06	Tue 4/4/06
25		**End-Product Demonstration**	**19 days?**	**Tue 4/4/06**	**Fri 4/28/06**
26		Demonstration Planning	6 days?	Tue 4/4/06	Tue 4/11/06
27		Faculty Advisor(s) Demonstration	4 days?	Tue 4/11/06	Fri 4/14/06
28		Client Demonstration	1 day?	Fri 4/21/06	Fri 4/21/06
29		Industrial Review Panel Demonstration	1 day?	Fri 4/28/06	Fri 4/28/06
30		**Project Reporting**	**179 days?**	**Tue 8/30/05**	**Fri 5/5/06**
31		Project Plan Development	19 days?	Tue 8/30/05	Fri 9/23/05
32		Project Plan Due	1 day?	Fri 9/23/05	Fri 9/23/05
33		Bound Project Plan Due	1 day?	Tue 10/11/05	Tue 10/11/05
34		End-Product Design Report Development	35 days?	Mon 9/26/05	Fri 11/11/05
35		End-Product Design Report Due	1 day?	Fri 11/11/05	Fri 11/11/05
36		Bound End-Product Design Report Due	1 day?	Wed 12/14/05	Wed 12/14/05
37		Project Poster Devlopment	37 days?	Mon 1/9/06	Tue 2/28/06
38		Project Poster Due	1 day?	Tue 2/28/06	Tue 2/28/06
39		Project Final Report Development	75 days?	Mon 11/14/05	Fri 3/31/06
40		Project Final Report Due	1 day?	Fri 3/31/06	Fri 3/31/06
41		Bound Project Final Report Due	1 day?	Wed 5/3/06	Wed 5/3/06
42		Weekly Email Reporting	154 days?	Tue 8/30/05	Fri 5/5/06



Project: May06-03
Date: Sat 4/29/06

Task		Progress		Summary		External Tasks	
Split		Milestone		Project Summary		External Milestone	

Deadline

Figure 9: Original Project Gantt Chart

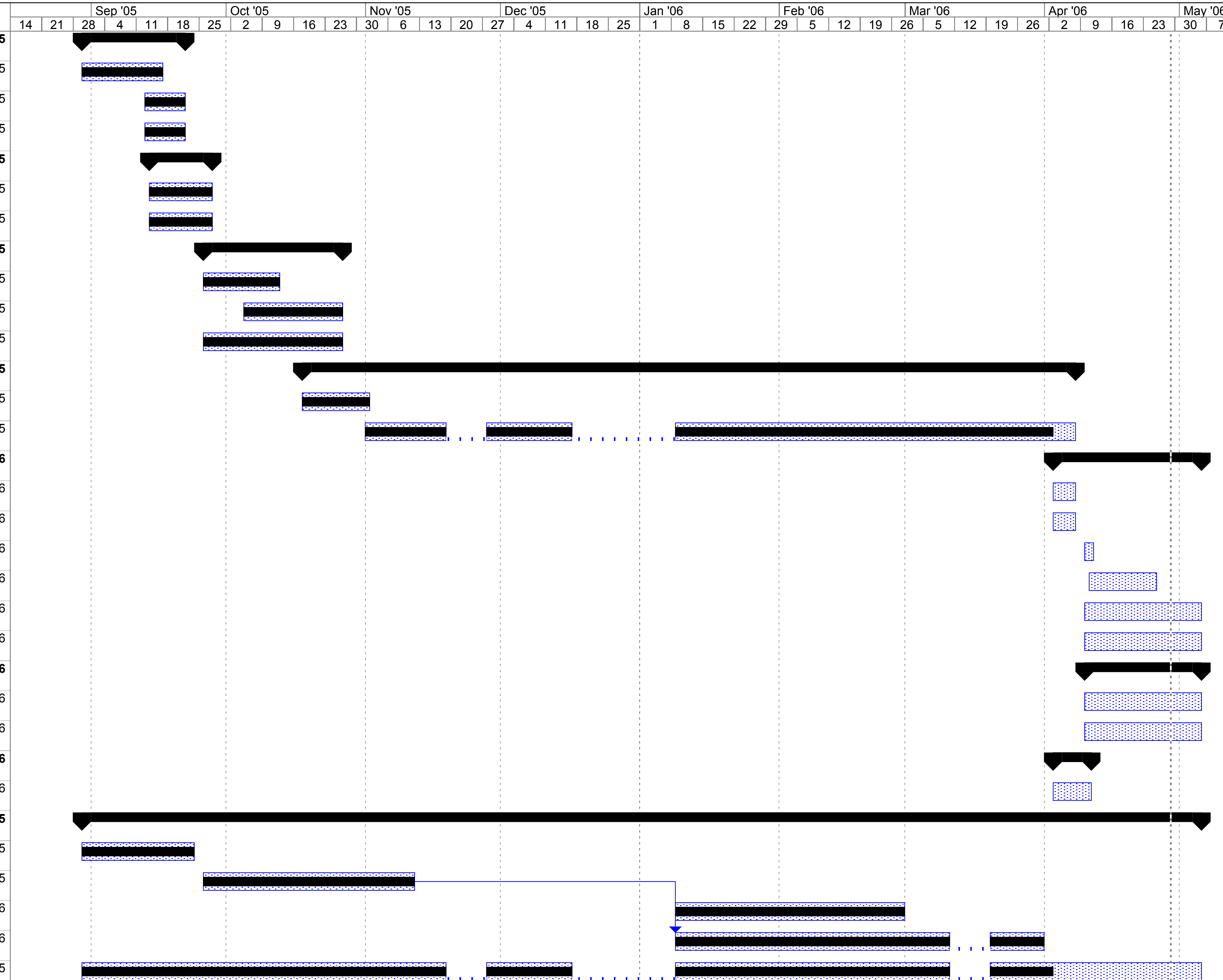
ID	Task Name	Duration	Start	Finish
1	**Problem Definition**	**17 days?**	**Tue 8/30/05**	**Wed 9/21/05**
2	Problem Definition Completion	14 days?	Tue 8/30/05	Fri 9/16/05
3	Confirmation of End-User(s) and End-Use(s)	7 days?	Tue 9/13/05	Wed 9/21/05
4	Review and Identification of Project Constraints	7 days?	Tue 9/13/05	Wed 9/21/05
5	**Technology Considerations and Selection**	**10 days?**	**Wed 9/14/05**	**Tue 9/27/05**
6	Review and Modification of Technologies	10 days?	Wed 9/14/05	Tue 9/27/05
7	Technology Selection	10 days?	Wed 9/14/05	Tue 9/27/05
8	**End Project Design**	**23 days?**	**Mon 9/26/05**	**Wed 10/26/05**
9	Review of Design Requirements	13 days?	Mon 9/26/05	Wed 10/12/05
10	Design Process	16 days?	Wed 10/5/05	Wed 10/26/05
11	Documentation of Design	23 days?	Mon 9/26/05	Wed 10/26/05
12	**End-Product Prototype Implementation**	**44 days?**	**Tue 10/18/05**	**Fri 12/16/05**
13	Identification of Prototype Limitations and Substitutions	11 days?	Tue 10/18/05	Tue 11/1/05
14	Implementation of Prototype End Product	29 days?	Tue 11/1/05	Fri 12/16/05
15	**End Product Testing**	**60 days?**	**Mon 1/9/06**	**Fri 3/31/06**
16	Test Planning	5 days?	Mon 1/9/06	Fri 1/13/06
17	Test Development	5 days?	Mon 1/9/06	Fri 1/13/06
18	Test Execution	5 days?	Mon 1/9/06	Fri 1/13/06
19	Client Testing	53 days?	Mon 1/16/06	Wed 3/29/06
20	Test Evaluation	55 days?	Mon 1/9/06	Fri 3/31/06
21	Documentation of Testing	55 days?	Mon 1/9/06	Fri 3/31/06
22	**End Product Documentation**	**21 days?**	**Tue 3/7/06**	**Tue 4/4/06**
23	Development of End-User Documentation	11 days?	Tue 3/7/06	Tue 3/28/06
24	Delopment of Maintenance and Support Documentation	6 days?	Tue 3/28/06	Tue 4/4/06
25	**End-Product Demonstration**	**6 days?**	**Tue 4/4/06**	**Tue 4/11/06**
26	Demonstration Planning	6 days?	Tue 4/4/06	Tue 4/11/06
27	**Project Reporting**	**179 days?**	**Tue 8/30/05**	**Fri 5/5/06**
28	Project Plan Development	19 days?	Tue 8/30/05	Fri 9/23/05
29	End-Product Design Report Development	35 days?	Mon 9/26/05	Fri 11/11/05
30	Project Poster Devlopment	37 days?	Mon 1/9/06	Tue 2/28/06
31	Project Final Report Development	55 days?	Mon 1/9/06	Fri 3/31/06
32	Weekly Email Reporting	154 days?	Tue 8/30/05	Fri 5/5/06

Project: May06-03
Date: Sat 4/29/06

Task · Split · Progress · Milestone · Summary · Project Summary · External Tasks · External Milestone · Deadline

Figure 10: Adjusted Project Gantt Chart

ID	Task Name	Duration	Start
1	**Problem Definition**	**17 days?**	**Tue 8/30/05**
2	Problem Definition Completion	14 days?	Tue 8/30/05
3	Confirmation of End-User(s) and End-Usei	7 days?	Tue 9/13/05
4	Review and Identification of Project Constr	7 days?	Tue 9/13/05
5	**Technology Considerations and Selection**	**10 days?**	**Wed 9/14/05**
6	Review and Modification of Technologies	10 days?	Wed 9/14/05
7	Technology Selection	10 days?	Wed 9/14/05
8	**End Project Design**	**23 days?**	**Mon 9/26/05**
9	Review of Design Requirements	13 days?	Mon 9/26/05
10	Design Process	16 days?	Wed 10/5/05
11	Documentation of Design	23 days?	Mon 9/26/05
12	**End-Product Prototype Implementation**	**124 days?**	**Tue 10/18/05**
13	Identification of Prototype Limitations and :	11 days?	Tue 10/18/05
14	Implementation of Prototype End Product	94 days?	Tue 11/1/05
15	**End Product Testing**	**25 days?**	**Mon 4/3/06**
16	Test Planning	5 days?	Mon 4/3/06
17	Test Development	5 days?	Mon 4/3/06
18	Test Execution	2 days?	Mon 4/10/06
19	Client Testing	11 days?	Tue 4/11/06
20	Test Evaluation	20 days	Mon 4/10/06
21	Documentation of Testing	20 days	Mon 4/10/06
22	**End Product Documentation**	**20 days?**	**Mon 4/10/06**
23	Development of End-User Documentation	20 days?	Mon 4/10/06
24	Development of Maintenance and Support	20 days?	Mon 4/10/06
25	**End-Product Demonstration**	**6.5 days?**	**Mon 4/3/06**
26	Demonstration Planning	6.5 days?	Mon 4/3/06
27	**Project Reporting**	**179 days?**	**Tue 8/30/05**
28	Project Plan Development	19 days?	Tue 8/30/05
29	End-Product Design Report Development	35 days?	Mon 9/26/05
30	Project Poster Devlopment	37 days?	Mon 1/9/06
31	Project Final Report Development	55 days?	Mon 1/9/06
32	Weekly Email Reporting	154 days?	Tue 8/30/05

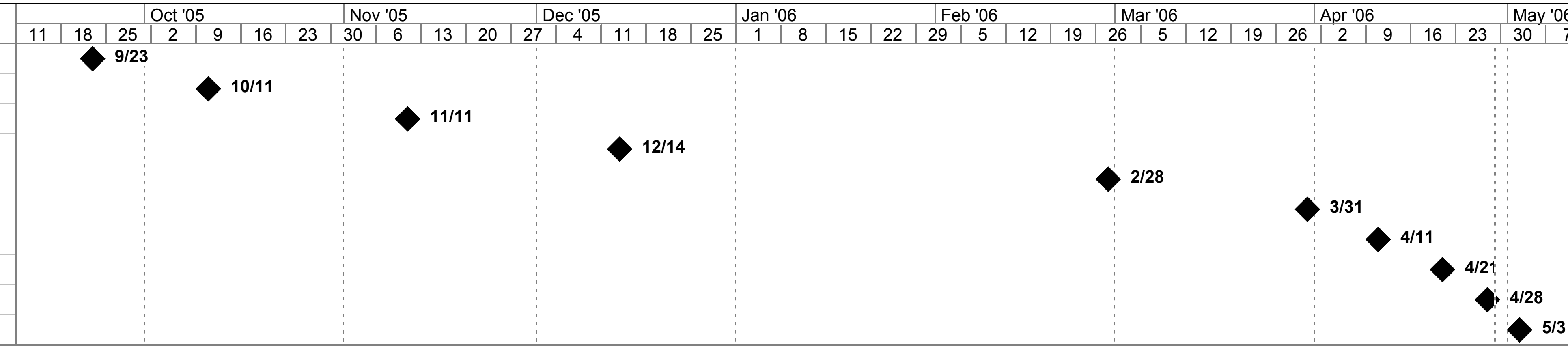


Project: May06-03
Date: Sat 4/29/06

Task | Progress | Summary | External Tasks | Deadline
Split | Milestone | Project Summary | External Milestone

Figure 11: Final Project Gantt Chart

ID		Task Name	Duration	Start	Finish	Predecessors
1		Project Plan Due	1 day?	Fri 9/23/05	Fri 9/23/05	
2		Bound Project Plan Due	1 day?	Tue 10/11/05	Tue 10/11/05	
3		End-Product Design Report Du	1 day?	Fri 11/11/05	Fri 11/11/05	
4		Bound End-Product Design Rep	1 day?	Wed 12/14/05	Wed 12/14/05	
5		Project Poster Due	1 day?	Tue 2/28/06	Tue 2/28/06	
6		Project Final Report Due	1 day?	Fri 3/31/06	Fri 3/31/06	
7		Faculty Advisor(s) Demonstratic	4 days?	Tue 4/11/06	Fri 4/14/06	
8		Client Demonstration	1 day?	Fri 4/21/06	Fri 4/21/06	
9		Industrial Review Panel Demon	1 day?	Fri 4/28/06	Fri 4/28/06	
10		Bound Project Final Report Due	1 day?	Wed 5/3/06	Wed 5/3/06	



Project: The Dream Green - Design Re
Date: Sat 4/29/06

Task | Split | Progress | Milestone | Summary | Project Summary | External Tasks | External Milestone | Deadline

Figure 12: Project Deliverables Gantt Chart - Unchanged throughout project

17 Project Evaluation

The project evaluation describes the milestones for the project and the criteria used to determine the success of the project. For each milestone, the team evaluated the completeness and relative success and assigned a score as shown in Table 9. The team used the scores for each individual milestone and used a weighting factor to calculate a final score for the project. A final score of 85% or more was considered a successful project.

Table 12: Evaluation Scores for Milestone Completion

Description	Score
Exceeded	100%
Met	100%
Almost Met	75%
Partially Met	50%
Did Not Attempt/Meet	0%

17.1 Project Definition

Description: The project was defined and the work of previous design teams was reviewed. The end-user and end-uses were confirmed. In addition, the project constraints were identified and confirmed.

Evaluation criteria: The project was defined thoroughly and based on the client's needs in reference to the final design.

Overall importance: 15%

Milestone score: Met - 100%

17.2 Technology Considerations and Selection

Description: The project team members initially reviewed the designs of prior teams. The design was modified to improve manufacturability and to better meet the needs of the client. Technologies were researched and evaluated based on their suitability.

Evaluation criteria: The technologies considered and selected are applicable to the intent and goals of the project.

Overall importance: 10%

Milestone score: Met - 100%

17.3 End-product Design

Description: Initially the design requirements were reviewed. The end-product design included the overall design process and component selection for the final product. The design changes were documented for future review and implementation.

Evaluation criteria: The end-product design process and design met the functional, financial, and manufacturing requirements of the project.

Overall importance: 20%

Milestone score: Met - 100%

17.4 End-product Implementation

Description: The end-product implementation is the construction of the end-product design, identification of the prototype limitations and substitutions of components. The end-product design was revised based on lessons learned during implementation.

Evaluation criteria: Prototype was constructed and conforms to the functional requirements and intended design.

Overall importance: 12%

Milestone score: Almost Met - 75%

17.5 End-product Testing

Description: The end-product testing was performed to identify and correct problems in the implementation of the design. The end-product testing consists of test planning, test development, test execution and client testing. Once the testing was completed, it was analyzed and documented.

Evaluation criteria: Prototype is mechanically functional, software has debugged, and user interface has been improved.

Overall importance: 15%

Milestone score: Partially Met - 50%

17.6 End-product Documentation

Description: The end-product documentation includes the development of a user manual for the hardware and software, drawings and bill of materials for manufacture, and the development of maintenance and support documents.

Evaluation criteria: End-product documentation includes information necessary for the construction, operation, and maintenance of the Dream Green.

Overall importance: 8%

Milestone score: Partially Met - 50%

17.7 End-product Demonstration

Description: The final prototype is presented to the client. Client input was used to finalize the design and to further evaluate the success of the project.

Evaluation criteria: End-product demonstration was useful and met the client's intended purpose of the project.

Overall importance: 8%

Milestone score: Met - 100%

17.8 Project Reporting

Description: The team developed several documents during the duration of this project. These documents include the project plan, end-product design report, project poster, and project final report. These documents were used to highlight the progress and revisions which led up to the completion of the project.

Evaluation criteria: Documentation met the necessary criteria and deliverables met established requirements.
Overall importance: 12%
Milestone score: Met - 100%

17.9 Final Project Score

The result of the project evaluation is an overall final project score of 85.5%. This score is calculated by adding the weighted average from each section. As defined above, this score exceeds the criteria necessary for a successful project.

18 Commercialization

Given that the manual Dream Green is currently for sale and has been for quite some time, it has been the goal from the start to make the automated Dream Green easily replicable for commercialization. It has been understood that the team had to make additional costs of automation minimal in order to allow a greater markup in price for the client, increasing his profit margins. There certainly would have been more expensive ways to accomplish the automation features of the Dream Green, but in selecting and purchasing parts cost was always considered in hopes that the client would be replicating the design.

19 Recommendations for Additional Work

Below is a list of features that could be added as future team projects:

1. Touch screen for improved and more user friendly interface.
2. Motor protection circuits to be implemented.
3. Convert motors to AC power.
4. User manual.
5. Computer interface for uploading/downloading new courses.

20 Lessons Learned

This section illustrates some of the lessons learned through the two-semesters designing and building a prototype Dream Green.

20.1 What went well

The team was able to successfully implement new technologies and refine existing ones to produce a working prototype for client evaluation. The team streamlined many previous designs, such as the end lift and motor positioning circuit, to make them easier to reproduce.

20.2 What did not go well

The team had a few set backs during this year. The first attempt to redesign the end lift was untimely unsuccessful and took much longer than expected because of the incremental design changes and difficulty with machining. Eventually a more efficient design was reached, but after missing the original deadlines.

20.3 What technical knowledge was gained

Because of there mechanical aspects in addition to the electrical aspects of this project the team gained valuable interdisciplinary design experience. The electrical and computer engineers were able to get a taste of fiction calculations, linear bearings, and power treads. The mechanical engineer was exposed to communications protocols and control circuits.

20.4 What non-technical knowledge was gained

In addition the technical skill the team gained communications experience with the design review, poster, project presentation, and industrial review panel. The team also gained knowledge in project planning and dealing with setbacks.

20.5 What would you do differently

The team would have liked to have met its original prototype deadline. This could have been accomplished by speeding up the design process and setting many smaller more achievable milestones.

21 Risk & Risk Management

The subsequent section will describe both risks and the risk management faced throughout the completion of the Dream Green project. The following section will discuss the anticipated potential risks and planned management, anticipated risks encountered and success in management, unanticipated risks encountered/attempts to manage and success, and the resultant changes in risk management made because of encountered unanticipated risks. Due to the nature of the project, there were many potential risks. This section explains how these risks were managed.

21.1 Anticipated potential risks and planned management

Many risks were anticipated to reduce the amount of problems further into the building of the Dream Green. Below is a list of the potential risks and the management steps taken to ensure that these risks would not result in major problems.

1. **Motor burnout** – The current was constantly monitored to ensure that the burnout current was not reached while the motors were being tested.
2. **Microcontroller burnout** – The controller boards were constantly monitored to ensure that no PIC was burned out.
3. **Power supply fire** – The power supply was turned off when no persons were present to ensure overheating would not take place.
4. **Bodily injury** – Due to the nature of the project, extra caution was taken to ensure that no bodily harm occurred while the Dream Green prototype was being constructed. Special precautions were taking in the form of gloves when handling potentially dangerous items. Many of the lubricants are dangerous if swallowed, to prevent any accidental ingestion regular hand washing took place.
5. **Loss of personal items and Dream Green specifications** – Many personal items were used throughout the construction of the Dream Green. To protect these items and the highly classified Dream Green information the door to the lab remained locked when there was no person present.
6. **Loss of team member** – There is always a possibility of a team member getting and internship even as a senior in college. To protect against this, the team shared all knowledge and information throughout the project.

21.2 Anticipated risks encountered and success in management

Many of the risks that were anticipated were avoided due to the precautions that were taken. When the current drawn by the motor reached a value around 3 amps the power supply was turned off. This prevented the accidental burnout of a motor. On one occasion, the microcontroller got warm. The work was immediately stopped and the microcontroller disconnected to ensure that the PIC was not burned out. No major bodily injury occurred during the building of the Dream Green prototype. A few cuts and scrapes were encountered from the sharp metal components of the prototype. Since the Dream Green lab was locked, no pertinent information was lost. There were also no break-ins to the Dream Green lab that would have lead to the loss of important Dream Green information or schematics.

21.3 Unanticipated risks encountered, attempts to manage and success

Although many of the risks were planned for ahead of time, not all risks could be avoided. Several unanticipated risks occurred while building the Dream Green prototype. These risks, the attempts to manage, and the success of this management are stated below.

1. **Breakage of important components** – There were times when important pieces of the Dream Green prototype were damaged. These pieces were damaged because they were caught under the Dream Green prototype. To manage this risk, there were several occasions in which the Dream Green lab was organized. By organizing the Dream Green lab these important pieces were not misplaced or damaged.
2. **Loss of important screws/bolts/nuts/etc.** – When the Dream Green lab was not organized many of the screws, nuts, bolts, and couplers were misplaced. Much time was wasted trying to find these parts. Once again, the Dream Green lab was organized to prevent the loss of these important items. By organizing the lab, these pieces were not lost and could be found easily. This reduced the time spent looking for parts.
3. **Low internet signal** – A risk that was not considered was the lack of wireless interned signal. By not having the internet, important data could not be acquired. This caused a risk in not completing the prototype on time. To manage this risk a ethernet cable was brought in. The ethernet cable provided a great signal and the internet was used to make many important decisions throughout the construction of the Dream Green prototype.

21.4 Resultant changes in risk management made because of encountered unanticipated risks

All unanticipated risks were dealt with on a prompt basis. These unanticipated risks did not become major problems due to the fast action in dealing with those particular risks. At the same time, other potential risks stemming from the unanticipated risk were also dealt with in a prompt manner. In addition, risks closely associated with the unanticipated risks were also thoroughly considered. By being proactive, many other unanticipated risks were avoided. The proactive approached saved much time and expense by dealing with problems before they happened. For instance, if a motor would have burned out it would have cost $30 to replace the motor and the time waiting for the motor would be lost. The anticipation of risks was a major factor in completing the Dream Green prototype on time.

22 Project Team Information

Listed below is the contact information for the student team members, faculty advisors, and client.

Team Members:

Ryan Emerson
Computer Engineering
2612 Aspen Rd. #3
Ames, IA 50010
515-290-4277
remerson@iastate.edu

David Goldberg
Electrical Engineering
4113 Frederiksen Ct
Ames, IA 50010
319-431-1593
goldberg@iastate.edu

David Moline
Electrical Engineering
3411 Polaris Dr.
Ames, IA 50010
515-231-9113
dmoline@iastate.edu

Mevan Vijithakumara
Mechanical Engineering
221 Sheldon #10
Ames, IA 50014
515-230-7724
mkumara@iastate.edu

Brian Wicks
Computer Engineering
3427 Polaris Dr. #2
Ames, IA 50010
515-231-2348
bwicks13@iastate.edu

Faculty Advisors:

Professor John Lamont
Office Phone: 515-294-3600
Home Phone: 515-292-5541
Fax: 515-294-6760
jwlamont@iastate.edu

Professor Ralph Patterson, III
Office Phone: 515-294-2428
Home Phone: 515-232-9933
Fax: 515-294-6760
repiii@iastate.edu

Client:

Charles Juel
Rt. 2
Stout, IA 50673
319-346-1608

23 Closing Summary

This team reevaluated the adjustment process and improved upon the internal components created from past teams. The team also created all modifications in as close to a "ready to manufacture" state as possible.

The team finalized the automation of the Dream Green modifiable putting surface. The team completed the current control systems necessary to automate these adjustments. The team made minor modifications to the work of previous teams and added necessary components. Such modifications included switching to a motor designed for robotics due to its greater torque handling capabilities. Added features include implementation of rotary encoders on all lifts, allowing the microcontroller to know the height each wedge. The team plans to present a working prototype to Mr. Juel, the client, by the project's end.

Appendix A - Code

dgmaster.c

```
/////////////////////////////////////////////////////////////////////////////
// Filename: dgmaster.c
//////////////////////////////////////////////////////////////////////////////
// Author:   Ryan Emerson
// Company:  Senior Design, Iowa State University
// Revision: 1.0
// Date:     9/14/05
//////////////////////////////////////////////////////////////////////////////

#include <p18f448.h>
#include <stdio.h>
#include <adc.h>
#include <delays.h>
#include <portb.h>
#include <reset.h>
#include <timers.h>
#include "can.h"
#include "UARTIntC.h"
#include "xlcd.h"
#include "keypad.h"
#include "dgcan.h"

////////////////////////
// Configuration Data //
////////////////////////
#pragma config OSC = HSPLL
#pragma config PWRT = ON
#pragma config BOR = ON
#pragma config WDT = OFF
#pragma config WDTPS = 128
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config CP0 = OFF
#pragma config CP1 = OFF
#pragma config WRTB = OFF
#pragma config WRTC = OFF
#pragma config WRTD = OFF
#pragma config EBTR0 = OFF
#pragma config EBTR1 = OFF
#pragma config EBTRB = OFF

//////////////////////////
// Constant Definitions //
//////////////////////////
#define NUM_OF_HOLES    18
#define MAX_MOTORS      18
#define MAX_PLAYERS     4
#define MAIN_MENU 0
#define GAME_OPT_0   1
#define GAME_OPT_1   2
#define GAME_OPT_2   3
#define GAME_OPT_3   4
#define GAME_0    5
#define GAME_1    6
#define GAME_2    7
#define GAME_3    8
#define MAINT_MENU   9
#define NEW_CRS_0 10
#define NEW_CRS_1 11
#define NEW_CRS_2 12
#define NEW_CRS_3 13
#define RND_CRS      14
```

```
#define SNEW_CRS_0   15
#define SNEW_CRS_1   16
#define SNEW_CRS_2   17
#define SNEW_CRS_3   18
#define MDL_MENU_0   19
#define MDL_MENU_1   20
#define SAVE_0 21
#define SAVE_1 22
#define SAVE_2 23
#define SAVE_3 24
#define INST_0 25

//////////////////////
// Macro Definitions //
//////////////////////
#define XLCDCursorOnBlinkOn()        XLCDCommand(0x0F)
#define XLCDCursorOnBlinkOff()       XLCDCommand(0x0E)
#define XLCDDisplayOnCursorOff()     XLCDCommand(0x0C)
#define XLCDDisplayOff()             XLCDCommand(0x08)
#define XLCDCursorMoveLeft()         XLCDCommand(0x10)
#define XLCDCursorMoveRight()        XLCDCommand(0x14)
#define XLCDDisplayMoveLeft()        XLCDCommand(0x18)
#define XLCDDisplayMoveRight()       XLCDCommand(0x1C)



////////////////////////
// Function Prototypes //
////////////////////////
unsigned char main_menu(void);
unsigned char game_opt0(void);
unsigned char game_opt1(void);
unsigned char game_opt2(void);
unsigned char game_opt3(void);
unsigned char game0(void);
unsigned char game1(void);
unsigned char game2(void);
unsigned char game3(void);
unsigned char maint_menu(void);
unsigned char new_crs0(void);
unsigned char new_crs1(void);
unsigned char new_crs2(void);
unsigned char new_crs3(void);
unsigned char rnd_crs(void);
unsigned char snew_crs0(void);
unsigned char snew_crs1(void);
unsigned char snew_crs2(void);
unsigned char snew_crs3(void);
unsigned char mdl_menu0(void);
unsigned char mdl_menu1(void);
unsigned char save_0(void);
unsigned char save_1(void);
unsigned char save_2(void);
unsigned char save_3(void);
unsigned char inst_0(void);


//////////////////////////
// Variable Declarations //
//////////////////////////
struct CANMessage TX_Message;
struct CANMessage RX_Message;
unsigned char state;
unsigned char num_players = 1;
unsigned char hole = 1;
unsigned isHole = 0;
unsigned char scores[MAX_PLAYERS] = {0, 0, 0, 0};
```

```
unsigned char newhole = 0;
unsigned char newpos = 0;
unsigned char newheight = 0;
unsigned char newcrsdone = 1;
unsigned char model[2] = {'2','0'};
unsigned char motor_count = 18;
unsigned char coursename[19] = {'L','I','T','T','L','E','
','A','U','G','U','S','T','A','\0','\0','\0','\0','\0'};
unsigned char crsnamelen = 14;
unsigned char saveflag = 0; //0 = loaded/saved, 1 = usr, 2 = rnd
unsigned char course[NUM_OF_HOLES][MAX_MOTORS] =
{{0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0},
 {0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0},
 {0, 4, 0, 0, 0, 0, 0, 0, 4, 0, 1, 0, 0, 2, 0, 0, 3, 0},
 {0, 4, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 0, 2, 0, 0, 3, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 3, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 4, 0, 0, 2, 0, 3, 0},
 {0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 4, 0, 0, 2, 0, 3, 0},
 {0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0},
 {0, 0, 0, 3, 0, 0, 0, 4, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0},
 {0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 2, 0, 0, 0},
 {0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 2, 0, 0, 3},
 {0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 3},
 {0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 3},
 {0, 0, 4, 0, 3, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 3},
 {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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

//////////////////////////
// ROM Declarations      //
//////////////////////////
rom unsigned char rndcoursename[19] = {'R','A','N','D','O','M','
','C','O','U','R','S','E','\0','\0','\0','\0','\0','\0'};
rom unsigned char rndcrsnamelen = 13;

//////////////////////////
// Functions             //
//////////////////////////
void XLCDDelay15ms (void)
{
   Delay10KTCYx(16);
    return;
}

void XLCDDelay4ms (void)
{
   Delay10KTCYx(60);
    return;
}

void XLCD_Delay500ns(void)
{
   Delay10TCYx(2);
   return;
}

void XLCDDelay(void)
    {
    int i;
    for(i=0;i<1000;i++)
        {
        Nop();
        }
    return;
    }
```

```
///////////////////////////////
// Interrupt Service Routine //
///////////////////////////////
void low_isr(void);// serial interrupt taken as low priority interrupt
#pragma code uart_int_service = 0x18
void uart_int_service(void)
{
   _asm  goto low_isr   _endasm

}
#pragma code

#pragma  interruptlow low_isr
void low_isr(void)
{
   UARTIntISR();
   KeypadISR();
}

#pragma interrupt HighISR
void HighISR(void)
{
   CANISR();
}

#pragma code highVector=0x08
void HighVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code  /* return to default code section */
//////////
// Main //
//////////
void main()
{
   unsigned char i;
   unsigned char chData;
   unsigned char last_state = -1;

   //Initialize CAN module
   CANInit();

   //Initalize Serial Controller
   UARTIntInit();
   mSetUARTRxIntLowPrior();
   mSetUARTTxIntLowPrior();

   //Initalize LCD
   ADCON1=0x07; //make PORTA digital as control portpins are from PORTA
   XLCDInit(); //initialize the LCD module

   state = MAIN_MENU;

   UARTIntPutChar('!');

   while(1)
   {

      // Update LCD Display
      //////////////////////
      if(state != last_state)
      {
         switch(state)
         {
```

```
            case MAIN_MENU: state = main_menu(); break;
            case GAME_OPT_0: state = game_opt0(); break;
            case GAME_OPT_1: state =  game_opt1(); break;
            case GAME_OPT_2: state =  game_opt2(); break;
            case GAME_OPT_3: state =  game_opt3(); break;
            case GAME_0: state =  state =  game0(); break;
            case GAME_1: state =  game1(); break;
            case GAME_2: state =  game2(); break;
            case GAME_3: state =  game3(); break;
            case MAINT_MENU: state =  maint_menu(); break;
            case NEW_CRS_0: state =  new_crs0(); break;
            case NEW_CRS_1: state =  new_crs1(); break;
            case NEW_CRS_2: state =  new_crs2(); break;
            case NEW_CRS_3: state =  new_crs3(); break;
            case RND_CRS:  state = rnd_crs(); break;
            case SNEW_CRS_0: state =  snew_crs0(); break;
            case SNEW_CRS_1: state =  snew_crs1(); break;
            case SNEW_CRS_2: state = snew_crs2(); break;
            case SNEW_CRS_3: state =  snew_crs3(); break;
            case MDL_MENU_0:  state = mdl_menu0(); break;
            case MDL_MENU_1:  state = mdl_menu1(); break;
            case SAVE_0:  state = save_0(); break;
            case SAVE_1:  state = save_1(); break;
            case SAVE_2:  state = save_2(); break;
            case SAVE_3:  state = save_3(); break;
            case INST_0:  state = inst_0(); break;
         }
      }

      // Check for CAN
      ////////////////
      if(CANRXMessageIsPending())
      {
         //New CAN, take it.
         RX_Message = CANGet();  //Get the message

         ///Is it from the master and is it for me?
         if(RX_Message.Address != 0xFF)
         {
            //Write to RS232 for fun
            UARTIntPutChar('@');
            UARTIntPutChar(TX_Message.Address);
            UARTIntPutChar('|');
            for(i=0; i < TX_Message.NoOfBytes; i++)
               UARTIntPutChar(TX_Message.Data[i]);
            UARTIntPutChar('|');
            UARTIntPutChar('\n');
            UARTIntPutChar('\r');

            //What kind of message is it?
            switch(RX_Message.Data[1])
            {
               case OVER_CURRENT:
               case OVER_TEMP:
               case OVER_TIME:
//                error(RX_Message.Data[0], RX_Message.Data[1]);
               break;
            }
         }
      }

      while(!vUARTIntStatus.UARTIntRxBufferEmpty)
      {
         UARTIntGetChar(&chData);
         UARTIntPutChar(chData); //echo

         if(chData == '\r' || chData == '\n')
```

```
        {
           for(i=0; i < TX_Message.NoOfBytes; i++)
              UARTIntPutChar(TX_Message.Data[i]);

           TX_Message.Address = 0xFF;
           TX_Message.NoOfBytes = 4;
           TX_Message.Ext = 0;
           TX_Message.Remote = 0;
           TX_Message.Priority = 2;
           CANPut(TX_Message);
           TX_Message.NoOfBytes = 0;
        }
        else
        {
           UARTIntPutChar(chData);
           TX_Message.Data[TX_Message.NoOfBytes] = chData;
           TX_Message.NoOfBytes++;
        }

     }
  }

  return;
}

unsigned char main_menu(void)
{
  unsigned char temp = 0;

  KeypadClose();
  XLCDReturnHome();
  XLCDClear();
  XLCDPutRomString((rom char *) "Main menu           ");
  XLCDPutRomString((rom char *) "2) Maintanence      ");
  XLCDPutRomString((rom char *) "1) Play Game        ");
  XLCDPutRomString((rom char *) "3) Instructions     ");
  XLCDCursorOnBlinkOff();

  KeypadOpen();
  while(temp != '1' && temp != '2' && temp != '3')
  {
     temp = KeypadGetChar();
  }
  KeypadClose();
  if(temp == '1') return GAME_OPT_0;
  if(temp == '2') return MAINT_MENU;
  if(temp == '3') return INST_0;

  return MAIN_MENU;
}

unsigned char game_opt0(void)
{
  unsigned char temp = 0;
  unsigned char i = 0;

  KeypadClose();
  XLCDReturnHome();
  XLCDClear();
  XLCDPutRomString((rom char *) "Game Options        ");
  XLCDPutRomString((rom char *) " ");
  for(i = 0; i < crsnamelen; i++)
  {
     XLCDPut(coursename[i]);
  }
  for(i = 0; i < (19 - crsnamelen); i++)
  {
```

```
      XLCDPutRomString((rom char *) " ");
   }
   XLCDPutRomString((rom char *) "1) Current Course   ");
   XLCDPutRomString((rom char *) "2) New Course       ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(temp != '1' && temp != '2' && temp != '*')
      temp = KeypadGetChar();
   KeypadClose();

   if(temp == '1') return GAME_OPT_1;
   if(temp == '2')
   {
      if(saveflag == 0) return NEW_CRS_0;
      else if(saveflag == 1 || saveflag == 2) return SAVE_1;
   }
   if(temp == '*') return MAIN_MENU;

   return MAIN_MENU;
}

unsigned char game_opt1(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Game Options        ");
   XLCDPutRomString((rom char *) "2. Play a Hole      ");
   XLCDPutRomString((rom char *) "1. Play Course      ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(temp != '1' && temp != '2' && temp != '*')
      temp = KeypadGetChar();
   KeypadClose();

   if(temp == '1')
   {
      isHole = 0;
      return GAME_OPT_3;
   }

   if(temp == '2')
   {
      isHole = 1;
      return GAME_OPT_2;
   }

   if(temp == '*') return GAME_OPT_0;

   return MAIN_MENU;
}

unsigned char game_opt2(void)
{
   unsigned char temp = 0;
   unsigned char count = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Game Options        ");
   XLCDPutRomString((rom char *) "                    ");
```

```
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Select Hole: ");
   XLCDCursorOnBlinkOn();

   hole = 0;
   KeypadOpen();
   while(temp != '#')
   {
      temp = KeypadGetChar();
      if(temp >= '0' && temp <= '9' && count < 2)
      {
         XLCDPut(temp);
         if(count == 1)
         {
            hole = hole * 10;
         }
         hole += temp - '0';
         count++;
      }
      if(temp == '*' && count > 0)
      {
         XLCDCursorMoveLeft();
         hole = hole / 10;
         count--;
      }
      if(temp == '*' && count == 0)
      {
         break;
      }
   }
   KeypadClose();

   if(temp == '*')
   {
      return GAME_OPT_1;
   }
   else
   {
      return GAME_OPT_3;
   }

   return MAIN_MENU;
}

unsigned char game_opt3(void)
{
   unsigned char temp = 0;
   unsigned char count = 0;

   num_players = 1;
   if(!isHole)
   {
      hole = 1;
   }

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Game Options        ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Players(1-4): ");
   XLCDCursorOnBlinkOn();

   KeypadOpen();
   while(temp != '#')
   {
```

```
      temp = KeypadGetChar();

      if(temp >= '1' && temp <= '4' && count == 0)
      {
         num_players = temp - '0';
         XLCDPut(temp);
         count++;
      }
      if(temp == '*' && count == 1)
      {
         XLCDCursorMoveLeft();
         count--;
      }
      if(temp == '*' && count == 0)
      {
         break;
      }
   }
   KeypadClose();

   if(temp == '*')
   {
      return GAME_OPT_1;
   }
   else
   {
      return GAME_2;
   }

   return MAIN_MENU;
}

unsigned char game0(void)
{
   int i;
   unsigned char temp = 0;
   unsigned char tempscore = 0;
   unsigned char count = 0;

   for(i=0; i < num_players; i++)
   {
      tempscore = 0;
      KeypadClose();
      XLCDReturnHome();
      XLCDClear();
      for(i = 0; i < crsnamelen; i++)
      {
         XLCDPut(coursename[i]);
      }
      for(i = 0; i < (20 - crsnamelen); i++)
      {
         XLCDPutRomString((rom char *) " ");
      }
      XLCDPutRomString((rom char *) "Player: ");
      XLCDPut( i + '1');
      XLCDPutRomString((rom char *) "           ");
      XLCDPutRomString((rom char *) "Hole: ");
      XLCDPut((hole / 10) + '0');
      XLCDPut((hole % 10) + '0');
      XLCDPutRomString((rom char *) "             ");
      XLCDPutRomString((rom char *) "Score: ");
      XLCDCursorOnBlinkOn();

      KeypadOpen();
      while(temp != '#')
      {
         temp = KeypadGetChar();
```

```
         if(temp >= '2' && temp <= '8' && count == 0)
         {
            tempscore = (temp - '0');
            XLCDPut(temp);
            count++;
         }
         if(temp == '*' && count == 1)
         {
            tempscore = 0;
            XLCDCursorMoveLeft();
            count--;
         }
      }
      scores[i] += tempscore;
      KeypadClose();
   }

   return GAME_1;
}

unsigned char game1(void)
{
   unsigned char i = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   for(i = 0; i < crsnamelen; i++)
   {
      XLCDPut(coursename[i]);
   }
   for(i = 0; i < (20 - crsnamelen); i++)
   {
      XLCDPutRomString((rom char *) " ");
   }
   XLCDPutRomString((rom char *) " P-1 ");
   XLCDPut((scores[0] / 100) + '0');
   XLCDPut(((scores[0] / 10) % 10) + '0');
   XLCDPut((scores[0] % 10) + '0');
   if(num_players >= 2)
   {
      XLCDPutRomString((rom char *) "   P-2 ");
      XLCDPut((scores[1] / 100) + '0');
      XLCDPut(((scores[1] / 10) % 10) + '0');
      XLCDPut((scores[1] % 10) + '0');
      XLCDPutRomString((rom char *) "  ");
   }
   else
   {
      XLCDPutRomString((rom char *) "            ");
   }

   XLCDPutRomString((rom char *) "After Hole ");
   XLCDPut((hole / 10) + '0');
   XLCDPut((hole % 10) + '0');
   XLCDPutRomString((rom char *) "       ");
   if(num_players >= 3)
   {
      XLCDPutRomString((rom char *) " P-3 ");
      XLCDPut((scores[3] / 100) + '0');
      XLCDPut(((scores[3] / 10) % 10) + '0');
      XLCDPut((scores[3] % 10) + '0');
      if(num_players == 4)
      {
         XLCDPutRomString((rom char *) "   P-4 ");
         XLCDPut((scores[4] / 100) + '0');
         XLCDPut(((scores[4] / 10) % 10) + '0');
```

```
         XLCDPut((scores[4] % 10) + '0');
         XLCDPutRomString((rom char *) "  ");
      }
      else
      {
         XLCDPutRomString((rom char *) "            ");
      }
   }
   else
   {
      XLCDPutRomString((rom char *) "                    ");
   }
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   if(isHole == 1 || hole == 18)
   {
      return GAME_3;
   }
   else
   {
      return GAME_2;
   }

   return MAIN_MENU;
}

unsigned char game2(void)
{
   unsigned char i = 0;

   hole++;
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   for(i = 0; i < crsnamelen; i++)
   {
      XLCDPut(coursename[i]);
   }
   for(i = 0; i < (20 - crsnamelen); i++)
   {
      XLCDPutRomString((rom char *) " ");
   }
   XLCDPutRomString((rom char *) "Loading Hole ");
   XLCDPut((hole / 10) + '0');
   XLCDPut((hole % 10) + '0');
   XLCDPutRomString((rom char *) "     ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Please Wait...      ");
   XLCDCursorOnBlinkOff();

   //temp - put course change here
   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return GAME_0;
}

unsigned char game3(void)
{
   unsigned char winner[4] = {0, 0, 0, 0};
   unsigned char score = 255;
   unsigned char tie = 1;
```

```
int i;

for(i = 0; i < num_players; i++)
{
   if(scores[i] < score)
   {
      score = scores[i];
      winner[0] = i + 1;
   }
}

for(i = 0; i < num_players; i++)
{
   if(scores[i] == score && winner[0] != (i + 1))
   {
      winner[tie] = i + 1;
      tie++;
   }
}

KeypadClose();
XLCDReturnHome();
XLCDClear();
for(i = 0; i < crsnamelen; i++)
{
   XLCDPut(coursename[i]);
}
for(i = 0; i < (20 - crsnamelen); i++)
{
   XLCDPutRomString((rom char *) " ");
}
XLCDPutRomString((rom char *) "  Congrats P-");
XLCDPut(winner[0] + '0');
if(tie > 1)
{
   XLCDPut(winner[1] + '0');
   if(tie > 2)
   {
      XLCDPut(winner[2] + '0');
      if(tie > 3)
      {
         XLCDPut(winner[3] + '0');
      }
      else
      {
         XLCDPutRomString((rom char *) "  ");
      }
   }
   else
   {
      XLCDPutRomString((rom char *) "    ");
   }
}
else
{
   XLCDPutRomString((rom char *) "      ");
}
if(tie == 1)
{
   XLCDPutRomString((rom char *) "WINNER!             ");
}
else
{
   XLCDPutRomString((rom char *) "WINNERS!            ");
}
XLCDPutRomString((rom char *) "    Score: ");
XLCDPut((score / 100) + '0');
```

```
   XLCDPut(((score / 10) % 10) + '0');
   XLCDPut((score % 10) + '0');
   XLCDPutRomString((rom char *) "       ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return MAIN_MENU;
}

unsigned char maint_menu(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Maintanence Menu    ");
   XLCDPutRomString((rom char *) "2) Change Model     ");
   XLCDPutRomString((rom char *) "1) New Course       ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(temp != '1' && temp != '2' && temp != '*')
      temp = KeypadGetChar();
   KeypadClose();

   if(temp == '1')
   {
      if(saveflag == 0) return NEW_CRS_0;
      else if(saveflag == 1 || saveflag == 2) return SAVE_1;
   }
   if(temp == '2') return MDL_MENU_0;
   if(temp == '*') return MAIN_MENU;

   return MAIN_MENU;
}

unsigned char new_crs0(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "New Course Menu     ");
   XLCDPutRomString((rom char *) "2) Set Your Own     ");
   XLCDPutRomString((rom char *) "1) Load New Course  ");
   XLCDPutRomString((rom char *) "3) Random Course    ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(temp != '1' && temp != '2' && temp != '3' && temp != '*')
   {
      temp = KeypadGetChar();
   }
   KeypadClose();
   if(temp == '1') return NEW_CRS_1;
   if(temp == '2') return SNEW_CRS_0;
   if(temp == '3') return RND_CRS;
   if(temp == '*') return MAIN_MENU;

   return MAIN_MENU;
}
```

```
unsigned char new_crs1(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "New Course        dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "Connect RS232 device");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "and Press #         ");
   XLCDCursorOnBlinkOff();

   //need connectivity stuff
   KeypadOpen();
   while(temp != '#' && temp != '*')
   {
      temp = KeypadGetChar();
   }
   KeypadClose();

   if(temp == '#') return NEW_CRS_2;
   if(temp == '*') return NEW_CRS_0;

   return MAIN_MENU;
}

unsigned char new_crs2(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "New Course      dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "Waiting for PC      ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Confirmation!       ");
   XLCDCursorOnBlinkOff();

   //need connectivity stuff
   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return NEW_CRS_3;
}

unsigned char new_crs3(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "New Course      dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "Loading,            ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Please Wait...      ");
   XLCDCursorOnBlinkOff();

   //need connectivity stuff
   KeypadOpen();
   while(KeypadGetChar() != '#');
```

```
   KeypadClose();

   saveflag = 0;
   return GAME_OPT_0;
}

unsigned char rnd_crs(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Random Course       ");
   XLCDPutRomString((rom char *) "Randomizing,        ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Please Wait...      ");
   XLCDCursorOnBlinkOff();

   //need course randomizing stuff
   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();


   saveflag = 2;
   return GAME_OPT_0;
}

unsigned char snew_crs0(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Set New Course      ");
   XLCDPutRomString((rom char *) "B is to Lower       ");
   XLCDPutRomString((rom char *) "A is to Raise       ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return SNEW_CRS_1;
}

unsigned char snew_crs1(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Set New Course      ");
   XLCDPutRomString((rom char *) "D changes Hole      ");
   XLCDPutRomString((rom char *) "C changes Position  ");
   XLCDPutRomString((rom char *) "# Finish            ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return SNEW_CRS_2;
}

unsigned char snew_crs2(void)
{
   unsigned char temp = 0;
   unsigned char chars = 0;
```

```
   unsigned char i = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Set New Course      ");
   XLCDPutRomString((rom char *) "Course Name         ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDCursorOnBlinkOn();

   KeypadOpen();
   while(temp != '#')
   {
      if(temp >= 'A' && temp <= 'Z')
      {
         coursename[chars] = temp;
         chars++;
      }
      if(temp == '*' && chars == 0)
      {
         return NEW_CRS_0;
      }
      if(temp == '*' && chars > 0)
      {
         chars--;
      }
   }
   KeypadClose();

   crsnamelen = chars + 1;

   if(chars == 0) return NEW_CRS_0;
   else
   {
      for(i = chars; i < 19; i++)
      {
         coursename[i] = '\0';
      }
   }

   return SNEW_CRS_3;
}

unsigned char snew_crs3(void)
{
   unsigned char temp = 0;

   if(newcrsdone)
   {
      newhole = 1;
      newpos = 1;
   }

   //load course
   newheight = course[newhole + 1][newpos + 1];

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Set New Course      ");
   XLCDPutRomString((rom char *) "Position: ");
   XLCDPut(newpos + '0');
   XLCDPutRomString((rom char *) "         ");
   XLCDPutRomString((rom char *) "Hole: ");
   XLCDPut(newhole + '0');
   XLCDPutRomString((rom char *) "             ");
   XLCDPutRomString((rom char *) "Height (0-4): ");
```

```
XLCDPut(newheight + '0');
XLCDPutRomString((rom char *) "      ");
XLCDCursorOnBlinkOff();

KeypadOpen();
while(temp != '#')
{
   if(temp == 'A')
   {
      if(newheight < 4)
      {
         newheight++;
      }

      course[newhole + 1][newpos + 1] = newheight;
      break;
   }
   if(temp == 'B')
   {
      if(newheight > 0)
      {
         newheight--;
      }

      course[newhole + 1][newpos + 1] = newheight;
      break;
   }
   if(temp == 'C')
   {
      if(newhole == 18)
      {
         newhole = 1;
      }
      else
      {
         newhole++;
      }

      break;
   }
   if(temp == 'D')
   {
      if(newpos == motor_count)
      {
         newpos = 1;
      }
      else
      {
         newpos++;
      }

      break;
   }
}
KeypadClose();

if(temp == '#')
{
   saveflag = 1;
   newcrsdone = 1;
   return GAME_OPT_0;
}

if(temp >= 'A' && temp <= 'D')
{
   return SNEW_CRS_3;
}
```

```
   return MAIN_MENU;
}

unsigned char mdl_menu0(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Model Menu      dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "2) 16' Dream Green  ");
   XLCDPutRomString((rom char *) "1) 12' Dream Green  ");
   XLCDPutRomString((rom char *) "3) 20' Dream Green  ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(temp != '1' && temp != '2' && temp != '3' && temp != '*')
   {
      temp = KeypadGetChar();
   }
   KeypadClose();

   if(temp == '*') return MAINT_MENU;

   if(temp == '1')
   {
      model[0] = '1';
      model[1] = '2';
      motor_count = 10;
   }

   if(temp == '2')
   {
      model[0] = '1';
      model[1] = '6';
      motor_count = 14;
   }

   if(temp == '3')
   {
      model[0] = '2';
      model[1] = '0';
      motor_count = 18;
   }

   return MDL_MENU_1;
}

unsigned char mdl_menu1(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "New Model      dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "course!             ");
   XLCDPutRomString((rom char *) "You must load a new ");
   XLCDPutRomString((rom char *) "Press #             ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(KeypadGetChar() != '#');
```

```
   KeypadClose();

   return NEW_CRS_0;
}

unsigned char save_0(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Your current course ");
   XLCDPutRomString((rom char *) "1) Save            ");
   XLCDPutRomString((rom char *) "has not been saved! ");
   XLCDPutRomString((rom char *) "2) Ignore          ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(temp != '1' && temp != '2' && temp != '3' && temp != '*')
   {
      temp = KeypadGetChar();
   }
   KeypadClose();
   if(temp == '1') return SAVE_1;
   if(temp == '2') return NEW_CRS_0;
   if(temp == '*') return MAIN_MENU;

   return MAIN_MENU;
}

unsigned char save_1(void)
{
   unsigned char temp = 0;

   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Save Course     dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "Connect RS232 device");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "and Press #         ");
   XLCDCursorOnBlinkOff();

   //need connectivity stuff
   KeypadOpen();
   while(temp != '#' && temp != '*')
   {
      temp = KeypadGetChar();
   }
   KeypadClose();

   if(temp == '#') return SAVE_2;
   if(temp == '*') return SAVE_0;

   return MAIN_MENU;
}

unsigned char save_2(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Save Course     dg");
   XLCDPut(model[0]);
```

```
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "Waiting for PC      ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Confirmation!       ");
   XLCDCursorOnBlinkOff();

   //need connectivity stuff
   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return SAVE_3;
}

unsigned char save_3(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Save Course     dg");
   XLCDPut(model[0]);
   XLCDPut(model[1]);
   XLCDPutRomString((rom char *) "Saving,             ");
   XLCDPutRomString((rom char *) "                    ");
   XLCDPutRomString((rom char *) "Please Wait...      ");
   XLCDCursorOnBlinkOff();

   //need connectivity stuff
   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   saveflag = 0;
   return NEW_CRS_0;
}

unsigned char inst_0(void)
{
   KeypadClose();
   XLCDReturnHome();
   XLCDClear();
   XLCDPutRomString((rom char *) "Instructions        ");
   XLCDPutRomString((rom char *) "# is Forward/Enter  ");
   XLCDPutRomString((rom char *) "At any time:        ");
   XLCDPutRomString((rom char *) "* is Back/Delete    ");
   XLCDCursorOnBlinkOff();

   KeypadOpen();
   while(KeypadGetChar() != '#');
   KeypadClose();

   return MAIN_MENU;
}
```

dgslave.c

```
////////////////////////////////////////////////////////////////////////////////
// Filename: dgslave.c
////////////////////////////////////////////////////////////////////////////////
// Author:   Ryan Emerson
// Company:  Senior Design, Iowa State University
// Revision: 1.0
// Date:        9/14/05
////////////////////////////////////////////////////////////////////////////////

#include <p18f248.h>
#include <stdlib.h>
#include <adc.h>
#include <delays.h>
#include <portb.h>
#include <reset.h>
#include <timers.h>
#include "can.h"
#include "UARTIntC.h"
#include "dgcan.h"

////////////////////////
// Configuration Data //
////////////////////////
#pragma config OSC = HSPLL
#pragma config PWRT = ON
#pragma config BOR = ON
#pragma config BORV = 27
#pragma config WDT = OFF
#pragma config WDTPS = 128
#pragma config STVR = ON
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config CP0 = OFF
#pragma config CP1 = OFF
#pragma config WRTB = OFF
#pragma config WRTC = OFF
#pragma config WRTD = OFF
#pragma config EBTR0 = OFF
#pragma config EBTR1 = OFF
#pragma config EBTRB = OFF

/////////////////
// I/O Mapping //
/////////////////
#define CURRENT_SENSE_0       PORTAbits.RA0
#define CURRENT_SENSE_1       PORTAbits.RA1
#define ID_BIT_0              PORTAbits.RA2
#define ID_BIT_1              PORTAbits.RA3
#define ID_BIT_2              PORTAbits.RA4
#define ID_BIT_3              PORTAbits.RA5
#define WEDGE0_ENCODER PORTBbits.RB0
#define WEDGE1_ENCODER PORTBbits.RB1
#define WEDGE0_HOME           PORTBbits.RB4
#define WEDGE1_HOME           PORTBbits.RB5
#define WEDGE0_END            PORTBbits.RB6
#define WEDGE1_END            PORTBbits.RB7
#define ENABLE_0              LATCbits.LATC0
#define ENABLE_1              LATCbits.LATC1
#define DIRECTION_0           LATCbits.LATC2
#define DIRECTION_1           LATCbits.LATC3
#define THERMAL_FLAG_0 PORTCbits.RC4
#define THERMAL_FLAG_1 PORTCbits.RC5
#define BRAKE_0                      LATCbits.LATC6
#define BRAKE_1                      LATCbits.LATC7


//////////////////////////
// Constant Definitions //
//////////////////////////
#define WEDGE_0               0
#define WEDGE_A     480
#define WEDGE_B     780
```

```
#define WEDGE_C     1080
#define WEDGE_D     1380
#define WEDGE_END      1680
#define TOLERANCE      24

#define MAX_CURRENT 20     //mA
#define SHUNT_VALUE 2200.0 //Ohms
#define CS_COFF        377.0  //µA/A.
#define V_REF       5.0          //Volts
#define MAX_TIME       26   //Seconds

///////////////////////
// Macro Definitions //
///////////////////////


/////////////////////////
// Function Prototypes //
/////////////////////////
void ISR(void);
float GetMotor0Current(void);
float GetMotor1Current(void);
void ProcessTarget0(void);
void ProcessTarget1(void);
void ProcessHome0(void);
void ProcessHome1(void);
void ProcessEnd0(void);
void ProcessEnd1(void);
void ProcessCurrentError0(unsigned char deciA);
void ProcessCurrentError1(unsigned char deciA);
void ProcessTempError0(void);
void ProcessTempError1(void);
void ProcessTimeError0(unsigned char deciSec);
void ProcessTimeError1(unsigned char deciSec);
void TurnOn_Motor0(void);
void TurnOff_Motor0(void);
void TurnOn_Motor1(void);
void TurnOff_Motor1(void);

///////////////////////////
// Variable Declarations //
///////////////////////////
unsigned long device_id;
unsigned char portB;
unsigned char motor0_on;
unsigned char motor1_on;
unsigned int wedge0_pos;
unsigned int wedge0_target;
unsigned int wedge1_pos;
unsigned int wedge1_target;
unsigned int motor0_time;
unsigned int motor1_time;
struct CANMessage TX_Message;
struct CANMessage RX_Message;


///////////////////////////////
// Interrupt Service Routine //
///////////////////////////////
void low_isr(void);// serial interrupt taken as low priority interrupt
#pragma code uart_int_service = 0x18
void uart_int_service(void)
{
       _asm   goto low_isr   _endasm

}
#pragma code

#pragma interruptlow low_isr
void low_isr(void)
{
       ISR();
}

#pragma interrupt HighISR
void HighISR(void)
```

```
{
        CANISR();
        ISR();
}

#pragma code highVector=0x08
void HighVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code  /* return to default code section */


//////////
// Main //
//////////
void main()
{
        ///////////////
        // Initalize //
        ///////////////
        TRISAbits.TRISA0 = 1; // Current Sense 0
        TRISAbits.TRISA1 = 1; // Current Sense 1
        TRISAbits.TRISA2 = 1; // ID Bit 0
        TRISAbits.TRISA3 = 1; // ID Bit 1
        TRISAbits.TRISA4 = 1; // ID Bit 2
        TRISAbits.TRISA5 = 1; // ID Bit 3
        TRISBbits.TRISB0 = 1; // Encoder 0
        TRISBbits.TRISB1 = 1; // Encoder 1
        TRISBbits.TRISB4 = 1; // Home 0
        TRISBbits.TRISB5 = 1; // Home 1
        TRISBbits.TRISB6 = 1; // End 0
        TRISBbits.TRISB7 = 1; // End 1 */
        TRISCbits.TRISC0 = 0; // Enable 0
        TRISCbits.TRISC1 = 0; // Enable 1
        TRISCbits.TRISC2 = 0; // Direction 0
        TRISCbits.TRISC3 = 0; // Direction 1
        TRISCbits.TRISC4 = 1; // Thermal Flag 0
        TRISCbits.TRISC5 = 1; // Thermal Flag 1
        TRISCbits.TRISC6 = 0; // Brake 0
        TRISCbits.TRISC7 = 0; // Brake 1

        device_id = 0;
        portB = PORTB;
        DIRECTION_0 = 0;
        DIRECTION_1    = 0;
        wedge0_pos = 0;
        wedge0_target = 0;
        wedge1_pos = 0;
        wedge1_target = 0;
        motor0_time = 0;
        motor1_time = 0;
        TurnOff_Motor0();
        TurnOff_Motor1();

        //Initialize CAN module
        CANInit();
        INTCON3bits.INT2IP = 1;
        RCONbits.IPEN = 1;                    //Enable priority interrupts
        INTCONbits.GIEH = 1;          //Enable all interrupts

        //Configure Timer0
        OpenTimer0( TIMER_INT_ON &
                              T0_16BIT &
                              T0_SOURCE_INT &
                              T0_PS_1_256 );
        INTCON2bits.TMR0IP = 1; //High priority

        OpenPORTB( PORTB_CHANGE_INT_ON &
                          PORTB_PULLUPS_OFF);
        INTCON2bits.RBIP = 0; //Low priority

        OpenRB0INT( PORTB_CHANGE_INT_ON &
                              RISING_EDGE_INT &
                              PORTB_PULLUPS_OFF );
```

```
	OpenRB1INT( PORTB_CHANGE_INT_ON &
			RISING_EDGE_INT &
			PORTB_PULLUPS_OFF );
	INTCON3bits.INT1IP = 0; //Low priority

	device_id = (ID_BIT_3 * 8) +
			(ID_BIT_2 * 4) +
			(ID_BIT_1 * 2) +
			(ID_BIT_0 * 1);

	device_id = (unsigned long)(device_id << 5);


	//Send ID CAN packet
	TX_Message.Address = (unsigned long) MASTER_ID | ACK;
	TX_Message.Data[0] = device_id;
	TX_Message.NoOfBytes = 1;
	TX_Message.Ext = 0;
	TX_Message.Remote = 0;
	TX_Message.Priority = 2;
	CANPut(TX_Message);

	// Home Wedge 0
	DIRECTION_0 = 1;
//	TurnOn_Motor0();
//	wedge0_target = 48;
//	while(!WEDGE0_HOME)	Nop();

	// Home Wedge 1
	DIRECTION_1 = 1;
//	TurnOn_Motor1();
//	while(!WEDGE1_HOME)	Nop();

	//////////////////////
	// Main Program Loop //
	//////////////////////
	while(1)
	{
		///////////////////
		// Check Position //
		///////////////////
		if(motor0_on)
		{
			// Overshoot protection, shouldn't be necessary
			//DIRECTION_0 = (wedge0_pos > wedge0_target ? 1 : 0);

			// Are we there yet?
			if(wedge0_pos < wedge0_target + TOLERANCE
				&& wedge0_pos > wedge0_target - TOLERANCE)
			if(wedge0_pos > 50)
				ProcessTarget0();
		}

		if(motor1_on)
		{
			// Overshoot protection, shouldn't be necessary
			DIRECTION_1 = (wedge1_pos > wedge1_target ? 1 : 0);

			// Are we there yet?
			if(wedge1_pos < wedge1_target + TOLERANCE
				&& wedge1_pos > wedge1_target - TOLERANCE)
				ProcessTarget1();
		}

		/////////////////////
		// Check for errors //
		/////////////////////
/*
		// Current
		//////////
		if(GetMotor0Current() > MAX_CURRENT)
			if(GetMotor0Current() > MAX_CURRENT)
				ProcessCurrentError0((unsigned char) GetMotor0Current());

		if(GetMotor1Current() > MAX_CURRENT)
			if(GetMotor1Current() > MAX_CURRENT)
```

```
                                ProcessCurrentError1((unsigned char) GetMotor1Current());
*/
                // Temp
                ///////
                if(THERMAL_FLAG_0)
                        ProcessTempError0();

                if(THERMAL_FLAG_1)
                        ProcessTempError1();

                ////////////////////////////
                // Check for CAN messages //
                ////////////////////////////
                if(CANRXMessageIsPending())
                {
                        //New CAN, take it.
                        RX_Message = CANGet(); //Get the message

                        //Is it from the master and is it for me?
                        if((RX_Message.Address & 0b1111100000) == device_id)
                        {
                                //What kind of message is it?
                                switch(RX_Message.Address & 0b0000011111)
                                {
                                        // Ping
                                        ///////
                                        case PING:
                                                //Send confirmation CAN packet
                                                TX_Message.Address = MASTER_ID | ACK;
                                                TX_Message.Data[0] = RX_Message.Data[0];
                                                TX_Message.Data[1] = RX_Message.Data[1];
                                                TX_Message.Data[2] = RX_Message.Data[2];
                                                TX_Message.Data[3] = RX_Message.Data[3];
                                                TX_Message.Data[4] = RX_Message.Data[4];
                                                TX_Message.Data[5] = RX_Message.Data[5];
                                                TX_Message.Data[6] = RX_Message.Data[6];
                                                TX_Message.Data[7] = RX_Message.Data[7];
                                                TX_Message.NoOfBytes = RX_Message.NoOfBytes;
                                                TX_Message.Ext = 0;
                                                TX_Message.Remote = 0;
                                                TX_Message.Priority = 2;
                                                CANPut(TX_Message);
                                                Delay1KTCYx(100);
                                        break;

                                        // Stop Wedges
                                        //////////////
                                        case STOP_WEDGES:
                                                TurnOff_Motor0();
                                                TurnOff_Motor1();
                                        break;

                                        // Move Wedge
                                        /////////////
                                        case MOVE_WEDGE:
                                                if(RX_Message.Data[0] == '0')
                                                {
                                                        switch(RX_Message.Data[1])
                                                        {
                                                                case '0': wedge0_target = WEDGE_0; break;
                                                                case 'A': wedge0_target = WEDGE_A; break;
                                                                case 'B': wedge0_target = WEDGE_B; break;
                                                                case 'C': wedge0_target = WEDGE_C; break;
                                                                case 'D': wedge0_target = WEDGE_D; break;
                                                        }

                                                        DIRECTION_0 = (wedge0_pos > wedge0_target ? 1 : 0);
                                                        TurnOn_Motor0();

                                                }
                                                else if(RX_Message.Data[0] == '1')
                                                {
                                                        switch(RX_Message.Data[1])
                                                        {
                                                                case '0': wedge1_target = WEDGE_0; break;
                                                                case 'A': wedge1_target = WEDGE_A; break;
```

```
                                                                        case 'B': wedge1_target = WEDGE_B; break;
                                                                        case 'C': wedge1_target = WEDGE_C; break;
                                                                        case 'D': wedge1_target = WEDGE_D; break;
                                                                }

                                                                DIRECTION_1 = (wedge1_pos > wedge1_target ? 1 : 0);
                                                                TurnOn_Motor1();
                                                        }
                                                break;
                                        }

                                }
                        }
                }

                return;
}

// Interrupt Service Routine
////////////////////////////////////////////////////////////////////////////////
void ISR(void)
{
        // Timmer 0
        if(INTCONbits.TMR0IE && INTCONbits.TMR0IF)
        {
                INTCONbits.TMR0IE = 0;
                INTCONbits.TMR0IF = 0;
                motor0_time += motor0_on;
                motor1_time += motor1_on;

                if(motor0_time > MAX_TIME)
                        ProcessTimeError0(motor0_time);

                if(motor1_time > MAX_TIME)
                        ProcessTimeError1(motor1_time);

                WriteTimer0(0);
                INTCONbits.TMR0IF = 0;
                INTCONbits.TMR0IE = 1;
        }

        //Encoder 0
        if(INTCONbits.INT0IE && INTCONbits.INT0IF)
        {
                INTCONbits.INT0IE = 0;
                INTCONbits.INT0IF = 0;
TX_Message.Address = MASTER_ID | 68;
CANPut(TX_Message);
                if(ENABLE_0 == 1)
                {
                        if(DIRECTION_0 == 1 && wedge0_pos > 0)
                                wedge0_pos++;
                        else if(DIRECTION_0    == 0 && wedge0_pos < 65536)
                                wedge0_pos++;
                        wedge0_pos += 1;
                }

                INTCONbits.INT0IF = 0;
                INTCONbits.INT0IE = 1;
        }

        //Encoder 1
        if(INTCON3bits.INT1IE && INTCON3bits.INT1IF)
        {
                INTCON3bits.INT1IE = 0;
                INTCON3bits.INT1IF = 0;
TX_Message.Address = MASTER_ID | 69;
CANPut(TX_Message);
                if(ENABLE_1 == 1)
                {
                        if(DIRECTION_1 == 1 && wedge1_pos > 0)
                                wedge1_pos--;
                        else if(DIRECTION_1    == 0 && wedge1_pos < 65536)
                                wedge1_pos++;
                }
```

```
                INTCON3bits.INT1IF = 0;
                INTCON3bits.INT1IE = 1;
        }

        //PORTB
        if(INTCONbits.RBIE && INTCONbits.RBIF)
        {
        INTCONbits.RBIF=0;

                //Wedge 0 Home
            if((portB ^ PORTB) & 0b00010000)
        {
                        //pin RB4 changed
                        TurnOff_Motor0();
                        wedge0_pos = 0;
                }
                //Wedge 1 Home
            if((portB ^ PORTB) & 0b00100000)
        {
                        //pin RB5 changed
                        TurnOff_Motor1();
                        wedge1_pos = 0;
                }
                //Wedge 0 End
            if((portB ^ PORTB) & 0b01000000)
        {
                        TurnOff_Motor0();
                        wedge0_pos = WEDGE_END;
                }
                //Wedge 0 Home
            if((portB ^ PORTB) & 0b10000000)
        {
                        TurnOff_Motor1();
                        wedge1_pos = WEDGE_END;
                }

                portB = PORTB;
    }

        return;
}

// Get Motor 0 Current
////////////////////////////////////////////////////////////////////////////////
float GetMotor0Current(void)
{
        unsigned int result;
        float voltage;

        //Configure A/D convertor
//      OpenADC( ADC_FOSC_32 &
//                      ADC_RIGHT_JUST &
//                      ADC_12_TAD,
//                      ADC_CH0 &
//                      ADC_INT_OFF, 15 );

        Delay10TCYx( 5 ); // Delay for 50TCY
        ConvertADC(); // Start conversion
//      while( BusyADC() ); // Wait for completion
        result = ReadADC(); // Read result
        CloseADC(); // Disable A/D converter

        //Convert to current
        voltage = ((float) result) / 1023.0 * V_REF;
        return (float) voltage / SHUNT_VALUE * CS_COFF; // I = V / R
}

// Get Motor 1 Current
////////////////////////////////////////////////////////////////////////////////
float GetMotor1Current(void)
{
        unsigned int result;
        float voltage;

        //Configure A/D convertor
//      OpenADC( ADC_FOSC_32 &
```

```
//                      ADC_RIGHT_JUST &
//                      ADC_12_TAD,
//                      ADC_CH0 &
//                      ADC_INT_OFF, 15 );

        Delay10TCYx( 5 ); // Delay for 50TCY
        ConvertADC(); // Start conversion
//      while( BusyADC() ); // Wait for completion
        result = ReadADC(); // Read result
        CloseADC(); // Disable A/D converter

        //Convert to current
        voltage = ((float) result) / 1023.0 * V_REF;
        return (float) voltage / SHUNT_VALUE * CS_COFF; // I = V / R
}


// Process Target 0
//////////////////////////////////////////////////////////////////////////////
void ProcessTarget0(void)
{
        TurnOff_Motor0();

        //Send confirmation CAN packet
        TX_Message.Address = MASTER_ID | MOVE_COMPLETE;
        TX_Message.NoOfBytes = 3;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '0';

        if(wedge0_pos >= 0 && wedge0_pos < WEDGE_A - TOLERANCE)
                TX_Message.Data[1] = '0';
        else if(wedge0_pos >= WEDGE_A - TOLERANCE &&
                 wedge0_pos < WEDGE_A + TOLERANCE)
                TX_Message.Data[1] = 'A';
        else if(wedge0_pos >= WEDGE_B - TOLERANCE &&
                 wedge0_pos < WEDGE_B + TOLERANCE)
                TX_Message.Data[1] = 'B';
        else if(wedge0_pos >= WEDGE_C - TOLERANCE &&
                 wedge0_pos < WEDGE_C + TOLERANCE)
                TX_Message.Data[1] = 'C';
        else if(wedge0_pos >= WEDGE_D - TOLERANCE &&
                 wedge0_pos <  WEDGE_D + TOLERANCE)
                TX_Message.Data[1] = 'D';
        else if(wedge0_pos >= WEDGE_D)
                TX_Message.Data[1] = 'E';
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        Delay1KTCYx(0);
        return;
}

// Process Target 1
//////////////////////////////////////////////////////////////////////////////
void ProcessTarget1(void)
{
        TurnOff_Motor1();

        //Send confirmation CAN packet
        TX_Message.Address = MASTER_ID | MOVE_COMPLETE;
        TX_Message.NoOfBytes = 3;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '1';

        if(wedge1_pos >= 0 && wedge1_pos < WEDGE_A - TOLERANCE)
                TX_Message.Data[1] = '0';
        else if(wedge1_pos >= WEDGE_A - TOLERANCE &&
                 wedge1_pos < WEDGE_A + TOLERANCE)
                TX_Message.Data[1] = 'A';
        else if(wedge1_pos >= WEDGE_B - TOLERANCE &&
                 wedge1_pos < WEDGE_B + TOLERANCE)
                TX_Message.Data[1] = 'B';
        else if(wedge1_pos >= WEDGE_C - TOLERANCE &&
                 wedge1_pos < WEDGE_C + TOLERANCE)
                TX_Message.Data[1] = 'C';
```

```
        else if(wedge1_pos >= WEDGE_D - TOLERANCE &&
                wedge1_pos < WEDGE_D + TOLERANCE)
                TX_Message.Data[1] = 'D';
        else if(wedge1_pos >= WEDGE_D)
                TX_Message.Data[1] = 'E';

        TX_Message.Data[1] = (wedge1_pos >> 8);
        TX_Message.Data[2] = (unsigned char) (wedge1_pos && 0xFF);
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        Delay100TCYx(0);
        return;
}


// Process Current Error 0
////////////////////////////////////////////////////////////////////////////////
void ProcessCurrentError0(unsigned char deciA)
{
        TurnOff_Motor0();

        //Send CAN Error Packet
        TX_Message.Address = MASTER_ID | CURRENT_ERROR;
        TX_Message.NoOfBytes = 3;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '0';
        TX_Message.Data[1] = deciA;
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        return;
}

// Process Current Error 1
////////////////////////////////////////////////////////////////////////////////
void ProcessCurrentError1(unsigned char deciA)
{
        TurnOff_Motor1();

        //Send CAN Error Packet
        TX_Message.Address = MASTER_ID | CURRENT_ERROR;
        TX_Message.NoOfBytes = 3;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '1';
        TX_Message.Data[1] = deciA;
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        return;
}

// Process Temperature Error 0
////////////////////////////////////////////////////////////////////////////////
void ProcessTempError0(void)
{
        TurnOff_Motor0();

        //Send CAN Error Packet
        TX_Message.Address = MASTER_ID | TEMP_ERROR;
        TX_Message.NoOfBytes = 2;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '0';
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        return;
}

// Process Temperature Error 1
////////////////////////////////////////////////////////////////////////////////
void ProcessTempError1(void)
{
        TurnOff_Motor1();

        //Send CAN Error Packet
        TX_Message.Address = MASTER_ID | TEMP_ERROR;
```

```
        TX_Message.NoOfBytes = 1;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '1';
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        return;
}

// Process Time Error 0
//////////////////////////////////////////////////////////////////////////////
void ProcessTimeError0(unsigned char deciSec)
{
        TurnOff_Motor0();

        //Send CAN Error Packet
        TX_Message.Address = MASTER_ID | TIME_ERROR;
        TX_Message.NoOfBytes = 2;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '0';
        TX_Message.Data[1] = deciSec;
        TX_Message.Priority = 0;
        CANPut(TX_Message);
        return;
}

// Process Time Error 1
//////////////////////////////////////////////////////////////////////////////
void ProcessTimeError1(unsigned char deciSec)
{
        TurnOff_Motor1();

        //Send CAN Error Packet
        TX_Message.Address = MASTER_ID | TIME_ERROR;
        TX_Message.NoOfBytes = 2;
        TX_Message.Ext = 0;
        TX_Message.Remote = 0;
        TX_Message.Data[0] = '1';
        TX_Message.Data[1] = deciSec;
        TX_Message.Priority = 2;
        CANPut(TX_Message);
        return;
}

// Turn On Motor 0
//////////////////////////////////////////////////////////////////////////////
void TurnOn_Motor0(void)
{
        BRAKE_0 = 0; //Disable brake
        Delay10TCYx(100);
        motor0_on = 1;
        ENABLE_0 = 1; //Turn motor on
        return;
}

// Turn Off Motor 0
//////////////////////////////////////////////////////////////////////////////
void TurnOff_Motor0(void)
{
        ENABLE_0 = 0; //Turn motor off
        motor0_time = 0; //Reset timmer
        motor0_on = 0; //Clear flag
        Delay1KTCYx(1); //Wait for spin down
        BRAKE_0 = 1; //Enable breake
        Delay1KTCYx(100);
        return;
}

// Turn On Motor 1
//////////////////////////////////////////////////////////////////////////////
void TurnOn_Motor1(void)
{
        BRAKE_1 = 0; //Disable brake
        Delay10TCYx(100);
        motor1_on = 1;
```

```
        ENABLE_1 = 1; //Turn motor on
        return;
}


// Turn Off Motor 1
////////////////////////////////////////////////////////////////////////////////
void TurnOff_Motor1(void)
{
        ENABLE_1 = 0; //Turn motor off
        motor1_time = 0; //Reset timmer
        motor1_on = 0; //Clear flag
        Delay1KTCYx(1); //Wait for spin down
        BRAKE_1 = 1; //Enable breake
        Delay1KTCYx(100);
}
```

dgcan.h

```
///////////////////////////////////////////////////////////////////////////////
// Filename: dgcan.h
////////////////////////////////////////////////////////////////////////////////
// Author:   Ryan Emerson
// Company:  Senior Design, Iowa State University
// Revision: 1.0
// Date:        9/14/05
////////////////////////////////////////////////////////////////////////////////


// Byte        9      8      7      6      5      4      3      2      1      0
//                    [--Destination--]   [--Packet Type--]


#define MASTER_ID (unsigned long)((unsigned long)0b010000 << 5)

#define ACK                          1 //Acknowledge a successfully received packet
#define PING                  2 //Ping (Send & Receive)
#define MOVE_WEDGE     3 //Command to move a wedge to a new positon
#define STOP_WEDGES           4 //Stop all wedges
#define MOVE_COMPLETE  5 //Can't find position error
#define POSITION_ERROR  6 //Request/Reply for current wedge position
#define CURRENT_ERROR         7 //Over-current error
#define TEMP_ERROR            8 //Over-temperature error
#define TIME_ERROR            9 //Over-time error
```

keyboard.h

```
//////////////////////////////////////////////////////////////////////////////
// Filename: keypad.h
//////////////////////////////////////////////////////////////////////////////
// Author:   Ryan Emerson
// Company:  Senior Design, Iowa State University
// Revision: 1.0
// Date:        9/14/05
//////////////////////////////////////////////////////////////////////////////

#ifndef __KEYPAD_H__
#define __KEYPAD_H__

#define COL_0 LATCbits.LATC0
#define COL_1 LATCbits.LATC1
#define COL_2 LATCbits.LATC2
#define COL_3 LATCbits.LATC3
#define ROW_0 PORTBbits.RB4
#define ROW_1 PORTBbits.RB5
#define ROW_2 PORTBbits.RB6
#define ROW_3 PORTBbits.RB7

#define KEYPAD_BUFFER_SIZE 8


void KeypadOpen(void);
void KeypadClose(void);
void KeypadClearCols(void);
void KeypadSetCols(void);
void KeypadPush(unsigned char chData);
unsigned char KeypadGetChar(void);
void KeypadISR(void);


#endif
```

keyboard.c

```
//////////////////////////////////////////////////////////////////////////////
// Filename: keypad.c
//////////////////////////////////////////////////////////////////////////////
// Author:   Ryan Emerson
// Company:  Senior Design, Iowa State University
// Revision: 1.0
// Date:        9/14/05
//////////////////////////////////////////////////////////////////////////////
#include <p18f448.h>
#include <stdio.h>
#include <delays.h>
#include "keypad.h"

unsigned char keypadBuffer;

unsigned char KeypadTable[4][4] = { {'D', '#', '0', '*'},
                                                                      {'C', '9', '8', '7'},
                                                                      {'B', '6', '5', '4'},
                                                                      {'A', '3', '2', '1'} };

void KeypadOpen(void)
{
        keypadBuffer = NULL;

        TRISCbits.TRISC0 = 0;
        TRISCbits.TRISC1 = 0;
        TRISCbits.TRISC2 = 0;
        TRISCbits.TRISC3 = 0;
        TRISBbits.TRISB4 = 1;
        TRISBbits.TRISB5 = 1;
        TRISBbits.TRISB6 = 1;
        TRISBbits.TRISB7 = 1;
```

```
        KeypadSetCols();

        INTCON2bits.RBPU = 1; //All PORTB pull-ups are disabled
        INTCONbits.RBIE = 1; //Enables the RB port change interrupt
        INTCONbits.RBIF = 0;
        INTCON2bits.RBIP = 0; //Low priority
}

void KeypadClose(void)
{
        INTCONbits.RBIE = 0;
        KeypadClearCols();
        keypadBuffer = NULL;
}

void KeypadClearCols(void)
{
        COL_0 = 0;
        COL_1 = 0;
        COL_2 = 0;
        COL_3 = 0;
        Delay100TCYx (2);
}

void KeypadSetCols(void)
{
        COL_0 = 1;
        COL_1 = 1;
        COL_2 = 1;
        COL_3 = 1;
        Delay100TCYx (2);
}

void KeypadPush(unsigned char chData)
{
        keypadBuffer = chData;
}

unsigned char KeypadGetChar(void)
{
        int i;
        unsigned char result = keypadBuffer;

        keypadBuffer = NULL;

        return result;
}


void KeypadISR(void)
{
        char temp;

        if(INTCONbits.RBIE && INTCONbits.RBIF)
        {
                INTCONbits.RBIE = 0;
                INTCONbits.RBIF = 0;

                if(ROW_0)
                {
                        KeypadClearCols();
                        COL_0 = 1;
                        Delay100TCYx (1);
                        if(ROW_0) KeypadPush(KeypadTable[0][0]);
                        KeypadClearCols();
                        COL_1 = 1;
                        Delay100TCYx (1);
                        if(ROW_0) KeypadPush(KeypadTable[0][1]);
                        KeypadClearCols();
                        COL_2 = 1;
                        Delay100TCYx (1);
                        if(ROW_0) KeypadPush(KeypadTable[0][2]);
                        KeypadClearCols();
                        COL_3 = 1;
                        Delay100TCYx (1);
                        if(ROW_0) KeypadPush(KeypadTable[0][3]);
```

```
                KeypadSetCols();
        }
        if(ROW_1)
        {
                KeypadClearCols();
                COL_0 = 1;
                Delay100TCYx (1);
                if(ROW_1) KeypadPush(KeypadTable[1][0]);
                KeypadClearCols();
                COL_1 = 1;
                Delay100TCYx (1);
                if(ROW_1) KeypadPush(KeypadTable[1][1]);
                KeypadClearCols();
                COL_2 = 1;
                Delay100TCYx (1);
                if(ROW_1) KeypadPush(KeypadTable[1][2]);
                KeypadClearCols();
                COL_3 = 1;
                Delay100TCYx (1);
                if(ROW_1) KeypadPush(KeypadTable[1][3]);
                KeypadSetCols();
        }
        if(ROW_2)
        {
                KeypadClearCols();
                COL_0 = 1;
                Delay100TCYx (1);
                if(ROW_2) KeypadPush(KeypadTable[2][0]);
                KeypadClearCols();
                COL_1 = 1;
                Delay100TCYx (1);
                if(ROW_2) KeypadPush(KeypadTable[2][1]);
                KeypadClearCols();
                COL_2 = 1;
                Delay100TCYx (1);
                if(ROW_2) KeypadPush(KeypadTable[2][2]);
                KeypadClearCols();
                COL_3 = 1;
                Delay100TCYx (1);
                if(ROW_2) KeypadPush(KeypadTable[2][3]);
                KeypadSetCols();
        }
        if(ROW_3)
        {
                KeypadClearCols();
                COL_0 = 1;
                Delay100TCYx (1);
                if(ROW_3) KeypadPush(KeypadTable[3][0]);
                KeypadClearCols();
                COL_1 = 1;
                Delay100TCYx (1);
                if(ROW_3) KeypadPush(KeypadTable[3][1]);
                KeypadClearCols();
                COL_2 = 1;
                Delay100TCYx (1);
                if(ROW_3) KeypadPush(KeypadTable[3][2]);
                KeypadClearCols();
                COL_3 = 1;
                Delay100TCYx (1);
                if(ROW_3) KeypadPush(KeypadTable[3][3]);
                KeypadSetCols();
        }

        INTCONbits.RBIF = 0;
        INTCONbits.RBIE = 1;
    }

    temp = PORTB;
    return;
}
```

Appendix B – Schematics

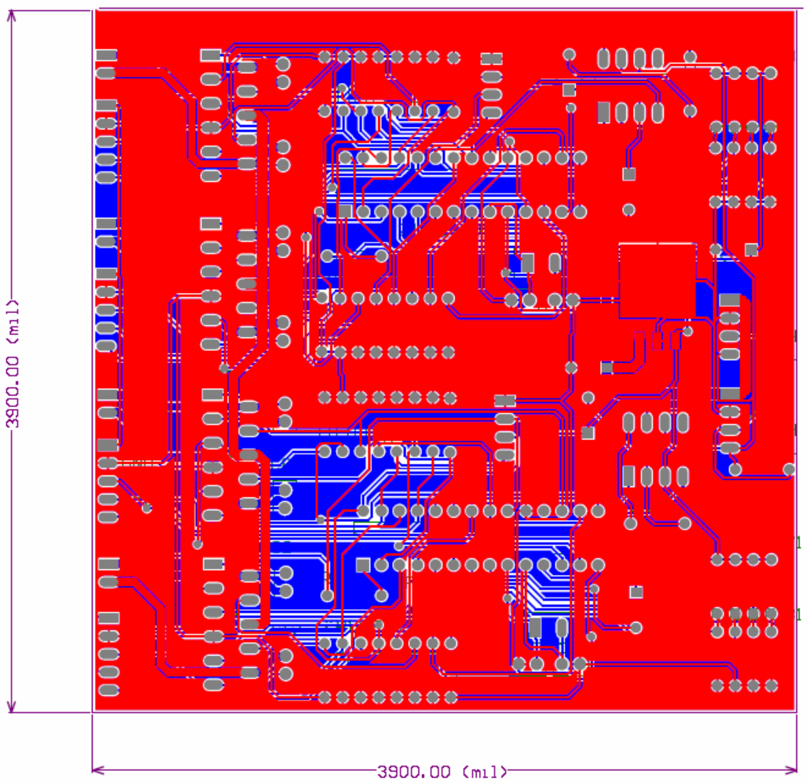


Figure 13: Motor Controller Circuit Board

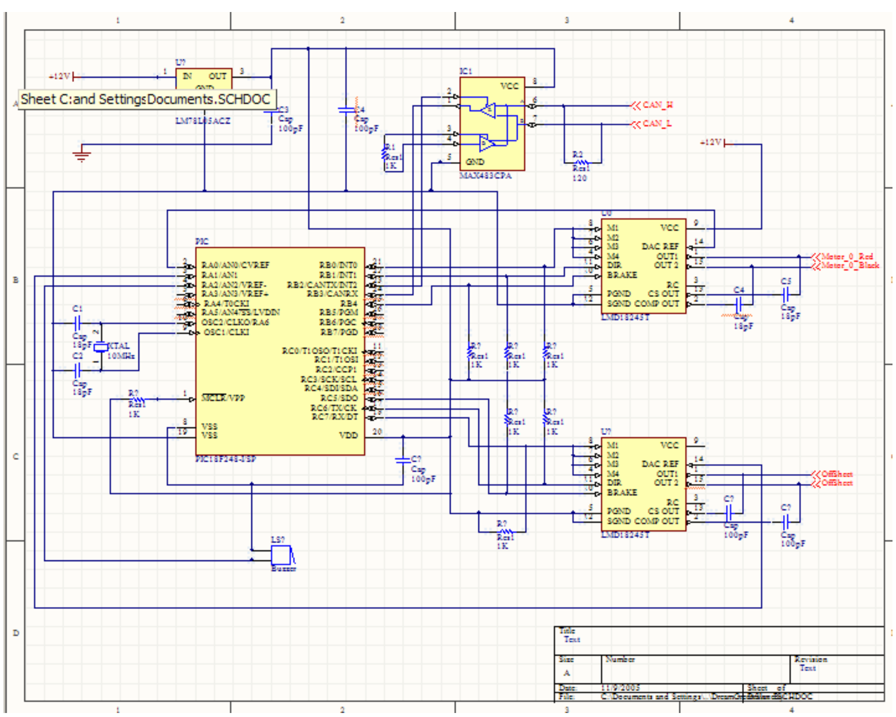


Figure 14: Motor Controller Circuit

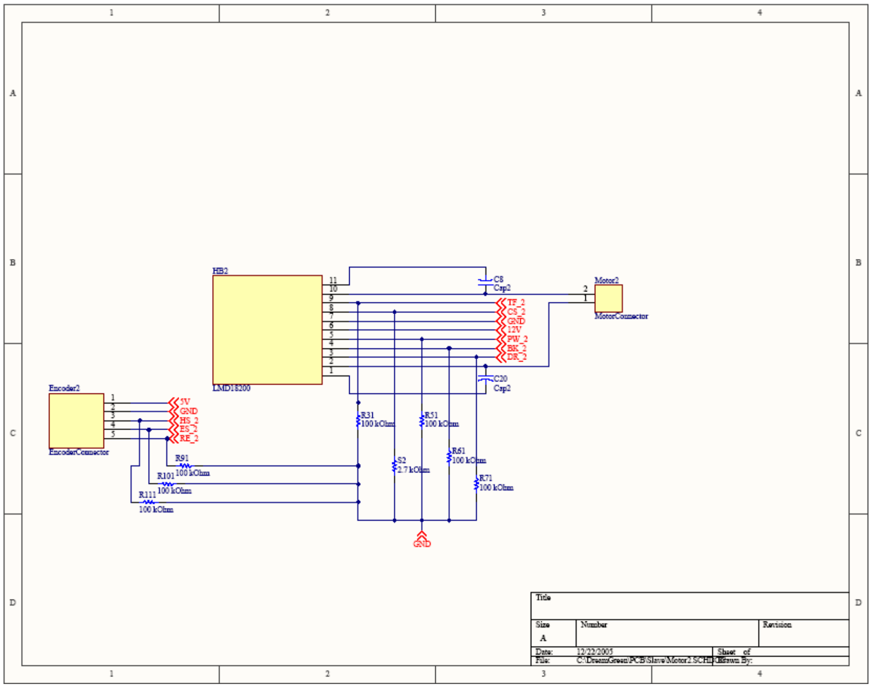


Figure 15: H-Bridge & Rotary Encoder

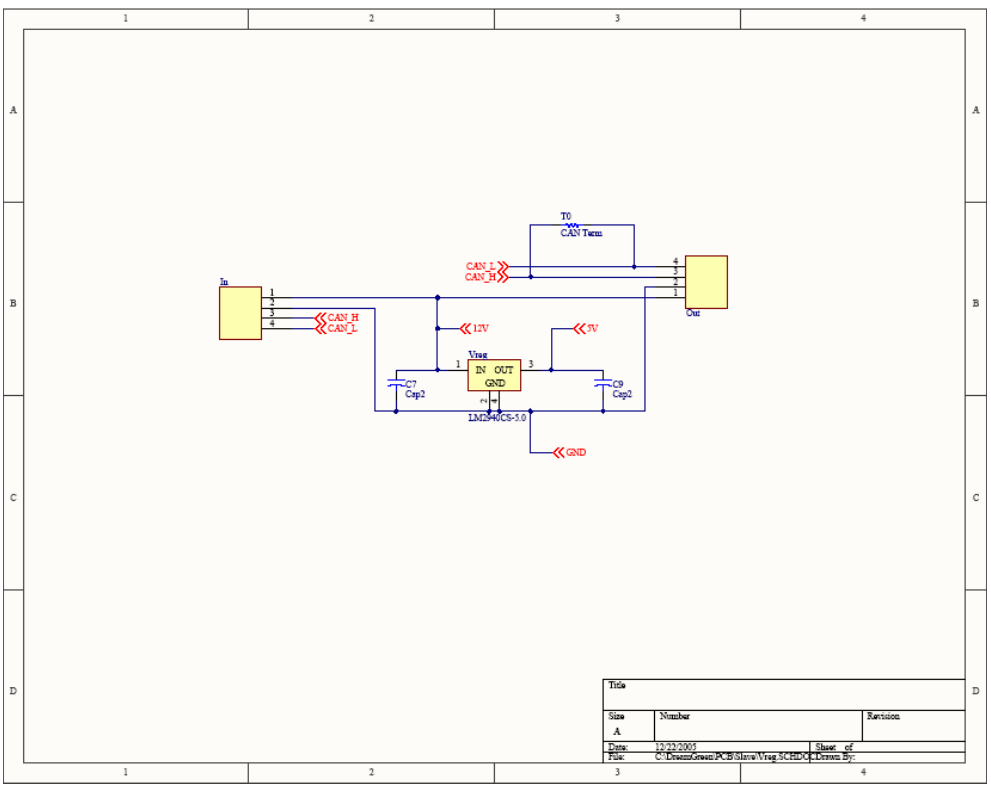


Figure 16: I/O Connection & Voltage Regulator

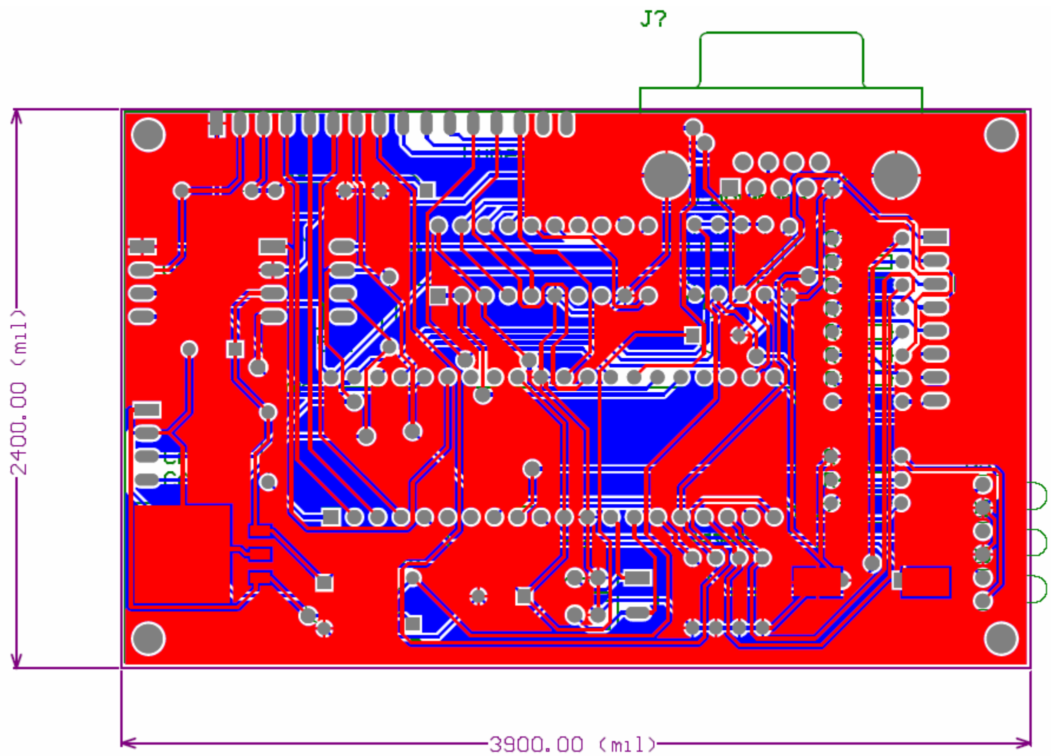


Figure 17: User Interface Circuit Card

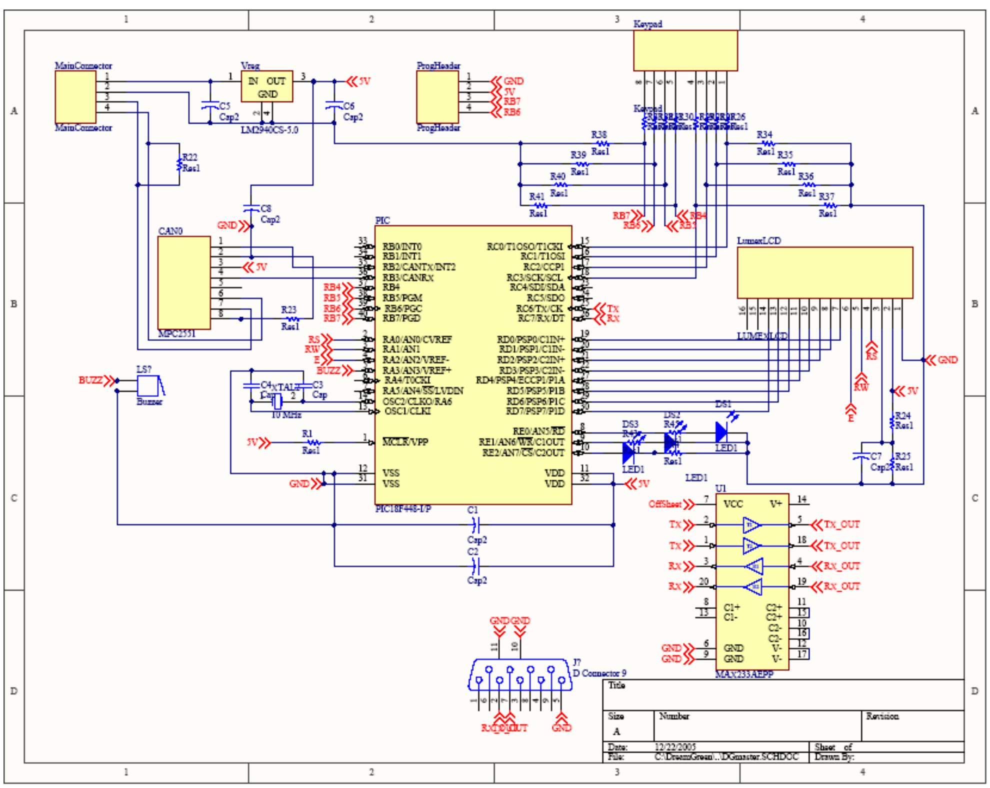


Figure 18: User Interface Circuit

Appendix C – Calculations

Table 13: Torque Calculations

L1	mu1	L2	mu2	FOS	Fa	d	p	mu3	Tr	Tl
35	0.35	36	0.15	2	35.3	0.375	0.0833	0.5	62.66	43.91
35	0.4	36	0.2	1.5	31.8	0.375	0.0833	0.5	56.44	39.55
35	0.4	36	0.4	2	56.8	0.250	0.0625	0.74	98.93	70.85
35	0.4	36	0.3	2	49.6	0.313	0.0714	0.74	106.52	78.51
35	0.4	36	0.08	2	33.76	0.375	0.0833	0.5	59.92	41.99

L1	Weight Applied
L2	Weight Applied and Wedge Weight
mu1	Wood - Wedge
mu2	Wedge - Guides
mu3	Screw - Threaded Insert

Thread/in	Length	Time	Rpm
20	7.500	0.7500	200
20	7.000	0.5700	245.614
16	7.000	0.5000	224