

IT 548
FALL 2004
MECHATRONICS ROBOT PROJECT

Submitted to:
Richard Helps
Information Technology
Brigham Young University



Prepared by
Ben Despain
Mark Eilertsen
Jeremiah K. Jones
Troy Tegeder
December 16, 2004

ABSTRACT

The purpose of this project was to create an autonomous robot to navigate a previously designated course. The robot was also given the task of seeking a ball of a given color while rejecting balls of other colors that may also be on the course. A variety of sensors, actuators, and mechanisms were used to complete the designated tasks and resulted in an autonomous robot able to complete most of the given tasks. The robot was unable to complete all tasks simultaneously.

TABLE OF CONTENTS

ABSTRACT.....	i
TABLE OF CONTENTS.....	ii
LIST OF TABLES AND FIGURES.....	iv

I. PROJECT DOCUMENTATION (1)

INTRODUCTION.....	1
OVERVIEW.....	1
The Competition.....	1
The Challenges.....	2
COMPONENTS ASSEMBLY.....	2
The Brick.....	2
Motors.....	2
Battery.....	3
Peripherals.....	3
<i>CMUcam</i>	3
<i>Ball Catcher</i>	4
<i>Phototransistor</i>	4
<i>Laser Range Finder</i>	5
<i>Encoder Modules</i>	5
CHASSIS.....	6
Concept Generation.....	6
Construction.....	6

MOBILITY.....	7
Hardware.....	7
Software.....	8
NAVIGATION.....	9
BALL SEEK AND IDENTIFICATION.....	10
BALL CAPTURE AND RELEASE.....	11
Design.....	11
Construction.....	11
GOAL LOCATE AND SCORE.....	13
UNCOMPLETED STRATEGIES.....	13
PROJECT ANALYSIS.....	15
SUGGESTIONS FOR FUTURE IMPROVEMENT.....	16

II. APPENDIXES (18)

POWER BUDGET.....	18
SOURCE CODE.....	18

III. BIBLIOGRAPHY (63)

LIST OF TABLES AND FIGURES

Title Page—Project Team Members	
Table 1—Competition Requirements.....	2
Figure 1—Battery.....	3
Figure 2—CMUcam.....	4
Figure 3—Encoders and Laser Range Finders.....	5
Figure 4—Original Robot Design.....	6
Figure 5—Final Robot Design.....	7
Table 2—Navigational Map.....	9
Figure 6—Cross Section of Ball Catcher and Release System.....	12
Figure 7—Ball Catcher and Release System.....	12

I. PROJECT DOCUMENTATION

INTRODUCTION

Mechatronics is the integration of mechanical, electrical, and computer systems. The purpose of this project was to implement knowledge gained about Mechatronics systems from the course to create an autonomous robot. The robot was given several tasks to complete including: navigating the lower-level of the specified course, traveling up and down slopes of two different grades, navigating the upper-level of the specified course, identifying balls of different color, capturing a ball of a specified color, returning home, and delivering the ball into the scoring zone.

This project used several different mechanical, electrical, and software devices and techniques in order to accomplish the previously defined tasks. Initially, each task was treated as a distinct problem to overcome, with the final goal of being able to combine all of the tasks into an autonomous device that could traverse the course while looking for, capturing, and delivering a ball of a specified color to the scoring zone.

OVERVIEW

The Competition

The primary goal of the competition is to design, build, and implement an autonomous robot capable of navigating a predefined course while looking for, capturing, and delivering balls of a specified color into a scoring zone. Each team was provided with the same equipment, resources, and timeframe. Scores were given based on the ability of the robot to perform certain tasks. The team with the overall highest score was presumably the winner of the competition.

The Challenges

The competition presented several tasks/problems that needed to be overcome:

Table 1 – Competition Requirements

1. Necessary components mounted to chassis for required activities
2. Chassis able to move while supporting wheels and other components
3. Navigation around the track
4. Ramp navigation
5. Ball color identification
6. Ball seeking
7. Ball acquisition
8. Return ball to scoring zone

Successful completion of each task resulted in a specified number of points being granted to the team. Extra points were awarded at the professor's discretion. Completion of all tasks resulted in additional bonus points.

COMPONENTS ASSEMBLY

The Brick

Developed by previous TAs and classes, “the brick” contains all of the basic functionality required to construct a robot. “The brick” includes a PC104 with a wireless card; a TS-5500 to provide serial ports, and Ethernet port, and ADC; a power regulation board; two H-bridges; a servo controller board; and an LCD. The brick is also provided with a stripped Linux Operating System and basic code to access the various components.

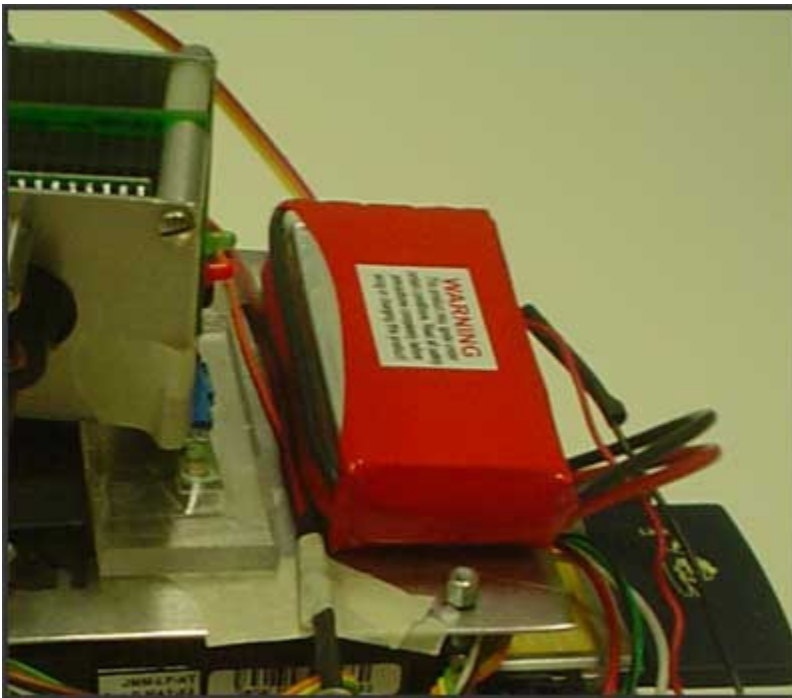
Motors

Two 1.5 V, DC motors with gear heads were provided as part of the Mechatronics kits. The motors provided mobility to both the right and left wheels of the robot. The motors were driven using a simulated Pulse Width Modulation (PWM) signal in order to control speed.

Battery

One 1500 mAh Lithium Polymer battery was provided for the competition. With all systems running, the battery lasted approximately 45 minutes. A complete power-budget calculation is provided on page 8 of the Appendix.

Figure 1—Battery

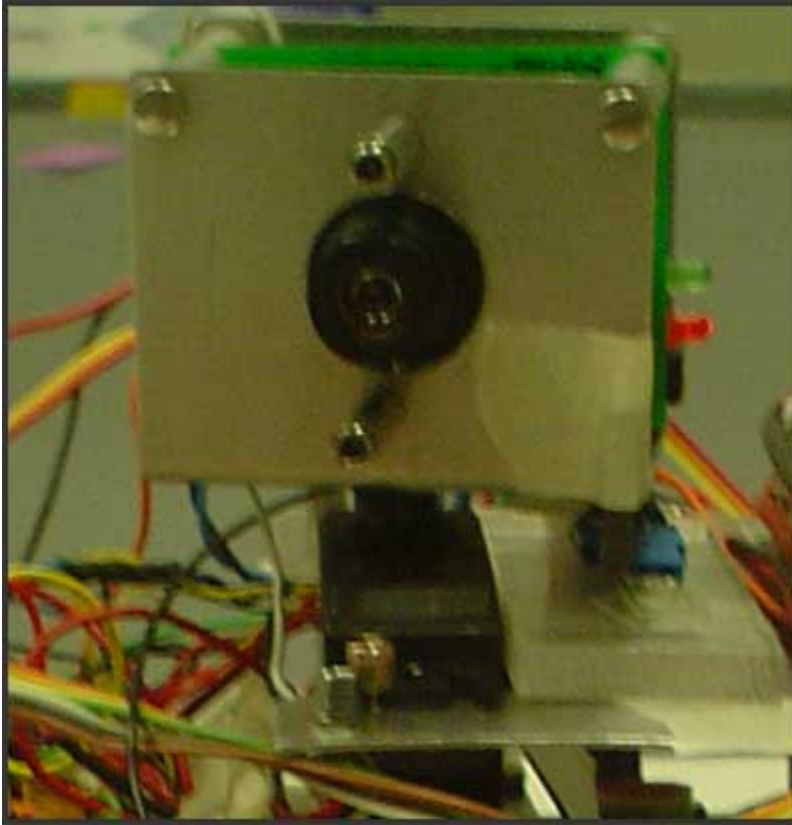


Peripherals

CMUcam

The CMUcam is a microcontroller-interfaced camera developed by Carnegie Mellon University. The camera can track colors, gather mean color data, and control a servo motor. The user can retrieve data through a RS-232 serial port.

Figure 2—CMUcam



Ball Catcher

The ball catcher and release mechanism was a custom-designed ball acquisition and expulsion system controlled by a DC motor. Further detail is provided in the “Ball Capture and Release” portion of this report on page 11.

Phototransistor

The phototransistor spectrally matches the frequency of an IR emitter. An analog output between 0 and 5V was outputted by the phototransistor based on the proximity to the emitter. Within this competition, the robot’s scoring zone was indicated by the emitter. The phototransistor assisted the robot in identifying its proximity to the scoring zone.

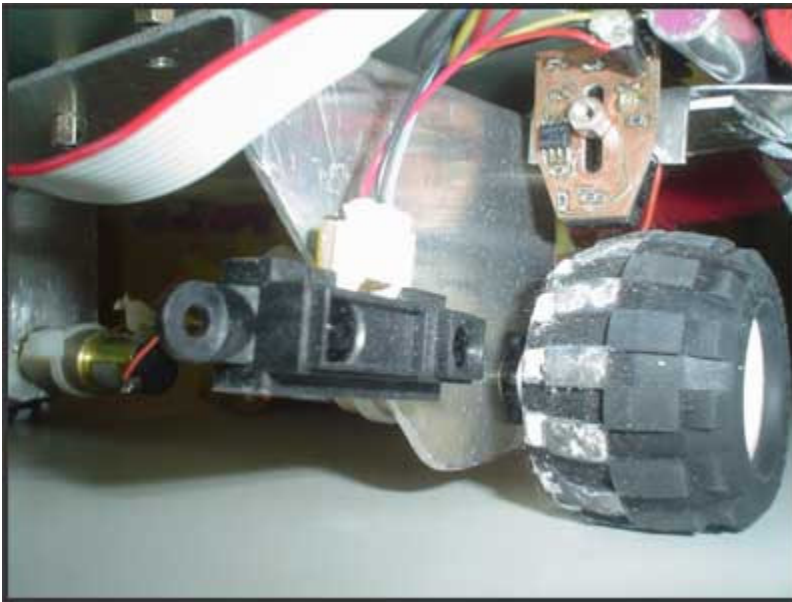
Laser Range Finder

Two laser range finders were mounted to the sides of the robot in order for the robot to determine its orientation on the track. The transmitters would send an analog signal to the robot indicating the distance to an object placed in front of the laser range finder. By sweeping the robot from side-to-side in front of a wall, the laser range finder was able to locate the nearest point, thus re-centering the robot and aligning it parallel with the surface of the wall.

Encoder Modules

In order to navigate the course, the robot used encoders on the wheels to track distance from one location to another. The encoders used one IR emitter and receiver on each wheel. The wheels had white stripes painted on the black rubber surface in order to indicate a change in light. The change in light would be detected and recorded by the robot in order to assist in tracking distance of travel.

Figure 3—Encoders and Laser Range Finders

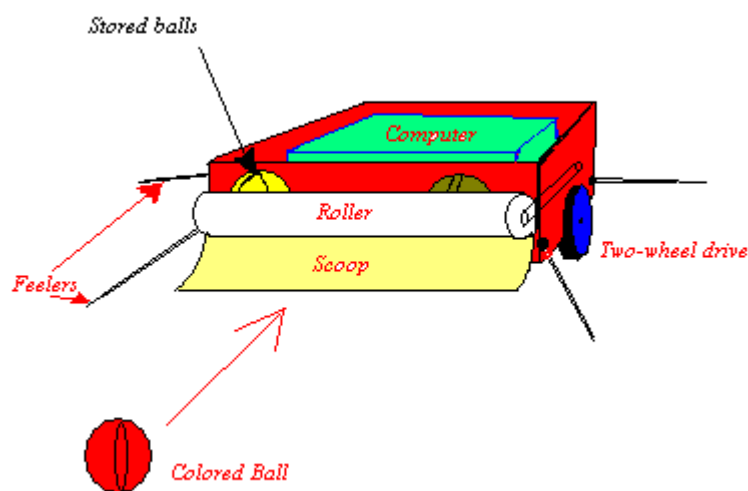


CHASSIS

Concept Generation

Once the competition rules were provided to the team, a concept generation session was held. During this session, all of the topics of the competition were discussed and several ideas formulated. A final concept was selected that the team felt would meet the needs of the competition. The design of the robot will be further discussed throughout this document. The original design is shown below.

Figure 4—Original Robot Design

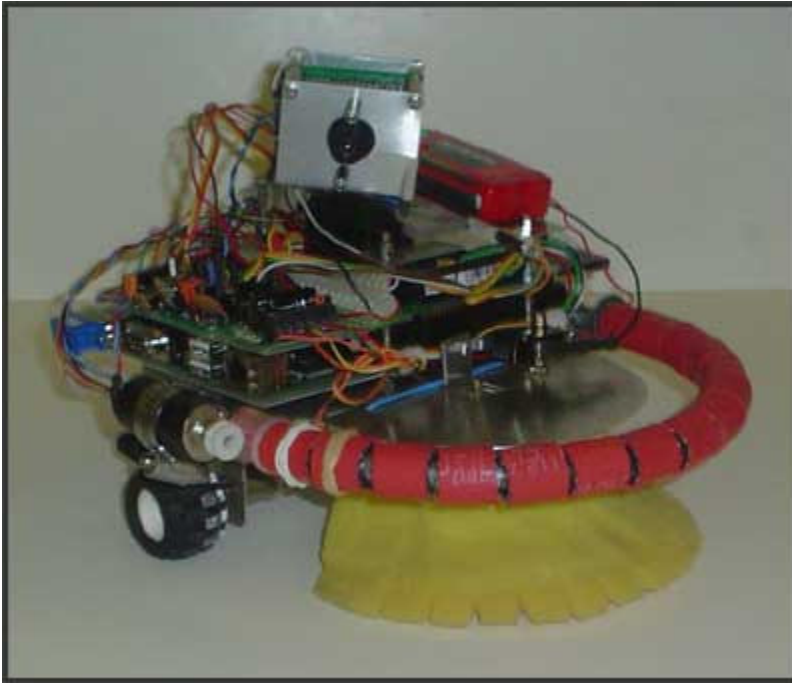


Construction

Once the design was finalized, the team began to construct the robot. The frame was constructed from sheet aluminum to provide light weight and easy manufacturing. In order to accommodate some of the extra components and solve design problems, the design was slightly modified from the original plan. Some of these changes included the

ball catching mechanism, materials for the construction, and the dimensions of the chassis. The final result of the design is shown below.

Figure 5—Final Robot Design



MOBILITY

Hardware

The movement of the robot was accomplished using two DC motors controlled through the H-bridge. The motors were mounted to the chassis using fasteners in the front plate of the motor. This provided a stable connection with very little affect from torque. Small, soft rubber tires were mounted to the motor shafts by an adapter manufactured by a team member. The motors for the robot were driven using the Digital Input/Output (DIO) on the brick. DIO1 was used to drive the left and right motors. This signal was then fed into the H-bridge to provide the motors with a protected source of power (to guard against back EMF).

The robot was designed to be small and maneuverable. The robot's axis of rotation was located very close to its geometric center, thus minimizing the space required for it to make a turn. The center of gravity of the robot was located just in front of the tires so that most of the weight of the robot was directly over the tires, thus improving traction. The dimensions of the robot measure almost the same in any direction in the horizontal plane, thus reducing the chances of getting caught or snagged on a corner.

Software

Software was written to communicate with DIO1 of the brick. The TS-5500 manual was used as a reference in order to determine what values needed to be sent to certain bits in order to configure and write-to the port. Please see the source code for the motors on page 52 of the Appendix for details on specific communication protocols used.

In order to modify the speed of the motors, a PWM signal was simulated using the processor cycles. For example, in order to slow down the motors, the enable bit of the appropriate DIO port would be set high, and would then be immediately set low for a variable amount of time. The amount of time would specify the speed of the motors. If full-speed was desired, then the enable bit was simply left high. Please refer to the `speedControl()` code on page 54 of the Appendix.

“Braking” code was also written in order to cause the motors to stop abruptly. The code was written in such a manner that the power of the “brakes” could be varied. Brakes were simulated by first sending an inverse signal to the motors for a variable amount of time, thus reversing the direction of the motors. The motors would then be turned to the off state, effectively stopping the motors, and then shutting them off. Please refer to the `motorsStopAll()` code on page 55 of the Appendix.

NAVIGATION

Before the navigation system is started, a map of the course was provided to the robot. The map was a textual representation of the course, focusing on the obstacles and objects of interest to complete the course. An example map is shown below, with syntax:

Table 2—Navigational Map

Each component of the map should be delimited by a tab and should follow this syntax: current position, next direction, object to north, distance to north, object to east, distance to east, object to south, distance to south, object to west, and distance to west.

```
HOME
STARTN      WALL 300   RAMP 0     WALL 300   DIP   200
WALL E      WALL 0     WALL 315   WALL 800   WALL 100
WALL E      WALL 0     OPEN 300   WALL 400   WALL 0
OPEN N      WALL 325   WALL 325   WALL 400   WALL 0
WALL E      WALL 0     OPEN 300   WALL 800   WALL 0
OPEN E      TRAP 100   OPEN 350   WALL 300   WALL 0
OPEN W      WALL 300   RAMP 75    WALL 300   HOME 25
HOME
```

The numbers are relative quantities to be calibrated and tested by the encoders that will be attached to the wheels. The map was loaded into the robot at run-time, and reads in one destination at a time. It will be assumed that the starting position and the opening position will be accurate. The robot will move towards the next destination according to the distance the map provides it with, and with appropriate feedback from the encoders on the wheels. Upon reaching the appropriate distance, the robot used a laser-range finder to scan the areas around it (only if the area was supposed to be a wall) and determined where the shortest point to that area was. It then turned itself until it was facing that point. For example, it read the distance to the wall in front of it; then turned slightly to the right to take another reading. If the current reading was greater than the previous reading, it turned back to the left, past its first reading until it reached slightly further. It then took another reading. It continued to move left as long as the current reading was less than the last reading. Once it finds a reading greater than the last, it turned back to face right, but less than the amount it just turned left, so that it was oriented between the last two points it

measured. It continued in this fashion until it had narrowed down upon the shortest distance between it and the wall in front of it (if it looped back and forth more than a variable number of times, then it would settle on the midpoint between the last two attempts and stop). This helped the robot to re-calibrate itself and ensure that it was not facing the wall at an angle. Please see the `lfCenter()` code on page 42 of the Appendix for more detail.

Once it had been calibrated, it would then continue onto the next destination and repeat the process described above. The robot would continue in this manner, until the “HOME” destination had been reached. Upon returning home, the robot would then attempt to locate the goal in order to drop a ball of at the scoring zone. The goal was located by trying to detect the array of infrared emitters that was on the goal. The robot used the same sweeping technique as described in the previous paragraphs in order to zero-in on the infrared array. Upon reaching a certain threshold (indicating the robot was very close to the emitters), the robot would then stop and begin to release the ball into the scoring vicinity.

BALL SEEK AND IDENTIFICATION

The main function of the CMUCam (camera) was to locate the ball. The camera was mounted on the top of the robot with a slight down slant in order to see the ground in front of the robot. The camera was programmed and calibrated to search for a ball of a specific color. Once the color had been found the camera would send a signal to the computer telling it that the color had been found. The robot would then cease all operations and determine the camera's orientation. The robot would then rotate to match that orientation and proceed to capture the ball.

BALL CAPTURE AND RELEASE

Design

One of the major problems with the designs of robots from past classes was the ball catcher mechanism. Most designs only have the catcher on the front end of the robot, so that if the robot is not properly aligned with the ball it would miss. The team's concept to solve this problem was to design a ball catcher that wrapped around the entire front end of the robot. This would allow the robot to catch the ball at any angle of attack, as well as capture balls that are up against a wall or in corners.

Construction

A round-stock aluminum rod that was bent in a half circle, was mounted to the front end of the chassis using hose clamps. A tension spring having an inner diameter equal to the outer diameter of the aluminum rod was used as a bearing that rotates about the aluminum rod. Frictionally set on this bearing were rubber links made from hose. The hose was cut into links so that it would easily articulate around the bend in the rod. A DC motor turned a belt that drove one of these rubber links. As this link was forced to rotate about the aluminum rod, the torque was translated to the bearing, which caused all of the rubber links to rotate about the rod. When any of these links came in contact with a ball, a torque was applied to the ball that caused it to roll up the ramp. The links continued to spin until the ball climbed high enough up the ramp to depress a limit switch, which indicated to the computer that the ball had been captured, and the DC motor stopped driving the links.

The limit switch consisted of a piece of aluminum sheet metal which hinged on the chassis of the robot. The sheet metal was cut in a semi-circle as well so that the ball would climb up the ramp and contact the sheet metal regardless of the angle at which the ball was captured. Located just above this piece of sheet metal was a push button switch so that when the sheet

metal was raised, the push-button switch was activated. A cross section of the system capturing a ball can be seen in figure 6. The mechanism released the ball simply by reversing the DC motor rotation when the goal was sensed.

Figure 6 - Cross Section of Ball Capture and Release Mechanism

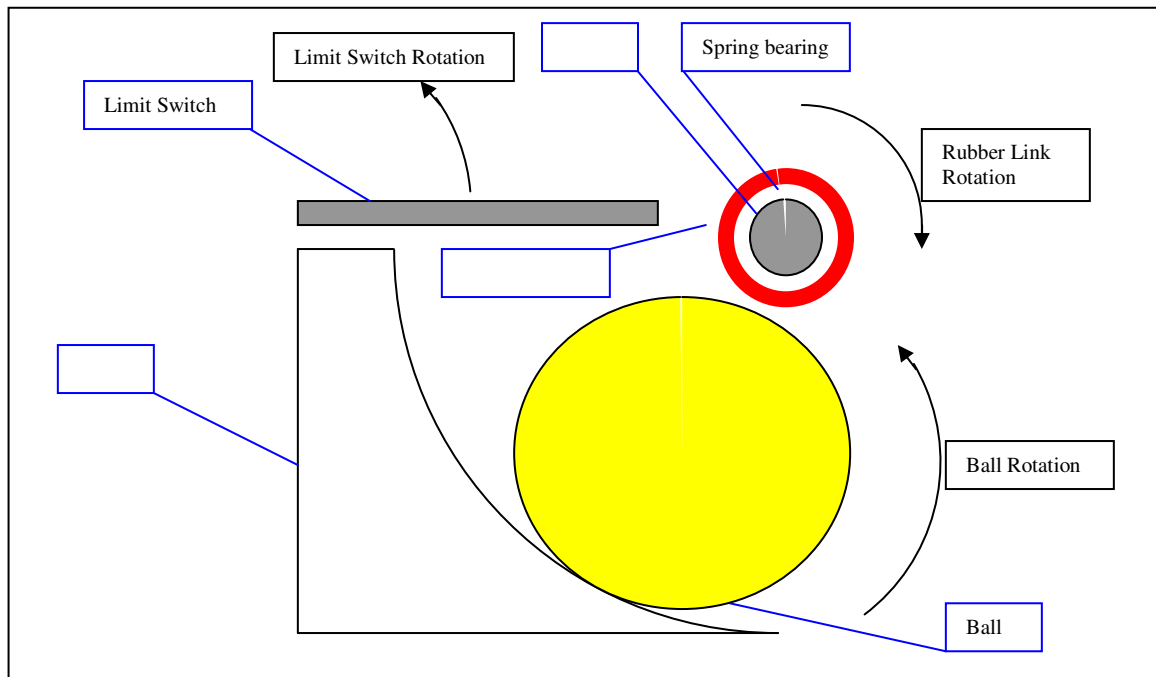
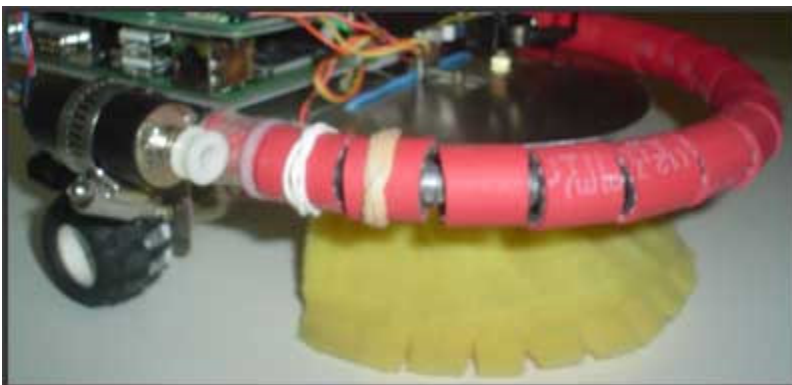


Figure 7—Ball Capture and Release Mechanism



GOAL LOCATE AND SCORE

The robot indicated completion of the course by locating the goal and releasing a captured ball into the goal box. The goal was constructed of a small wooden box with an opening on the front for holding balls and an array of infrared emitters placed on top of the goal box to assist the robot in finding the scoring zone.

The final destination in the course map would always point the robot in the general direction of where the goal was supposed to be placed. The robot would then activate a phototransistor that would detect the infrared signals being emitted by the infrared array. An algorithm would then control the movements of the robot, causing it to move in a sweeping motion from right to left in front of the scoring zone. It would be constantly seeking out the strongest signal from the infrared sensors. Each time the strong point was located, the robot would then move forward and sweep for the strongest signal once again. The robot would continue in this manner until the signal had reached a previously calibrated threshold, at which point the robot was most likely directly in front of the scoring zone. The robot would then cease movement, and would activate the ball-release mechanism by turning the motor in reverse direction of what was used to capture the ball. The motor would run for 8 seconds (a time calibrated during testing), which gave enough time for the ball to be released into the scoring zone. For detail about the algorithm used to locate the goal, please see the `psFind()` and `ps_goToDec()` functions on page 40 of the source code.

UNCOMPLETED STRATEGIES

Several tasks were left undone by the time the final competition came. This is not to say that there weren't many tasks that were completed successfully and satisfactorily as is noted throughout this report. The strategies left incomplete were mainly concerned with the CMU camera and its integration into the whole system. Also, each of the constituent tasks the robot

was able to complete needed to be integrated into the main program in order to implement each task in sequence without manual input.

At the time of the competition the CMU camera code was very effective at identifying balls of certain colors and tracking them, but lacked the ability to specify where the ball was located. Linking the ball location described by the CMU camera to the robot's orientation would have required inputting the CMU camera servo heading information into the navigation system and driving the wheels to move according the location given by the CMU camera. A Hall Effect sensor was used to center the CMU camera according to the direction of the robot to ensure correct orientation with respect to the direction the camera pointed as it tracked the ball. This system was in place and physically operative at the time of the competition but lacked the programming necessary to integrate it into the robot's main program.

Finally, each task performed needed to be combined into one command. This would have allowed the team to simply startup a program that would allow the robot to autonomously make decision on what actions to perform based on information received by the input devices. This would have entailed running the navigation routine and at each checkpoint or "waypoint" having the robot automatically search for a ball of specified color. Subsequently, when the robot had found the ball, it would implement the ball catch mode and move toward the ball and retrieve it. Once this task was completed it would then automatically move back to navigating the course and finally reach the goal. Then, upon reaching the goal the robot would automatically initiate ball release mode and release the ball into the goal. This, then, would have been followed by the robot entering into a second map mode in which it would move toward a ramp and search for balls on the upper level of the course. It would have completed the upper level in much the same way it completed the lower level course. The main program to run the robot was already set up to easily sequence these events, but the team was unable to finish synchronizing the camera motion with that of the robot's navigational system.

PROJECT ANALYSIS

Most of the tasks outlined in the project were completed successfully. The robot was able to navigate the course autonomously, and could identify, capture, and release a ball. The robot was able to move up and down both a shallow and steep ramp, and was able (with some assistance) to navigate the upper level of the track as well. The robot was also able to use the phototransistor to locate the goal and release a ball near the scoring zone. The robot also had the fastest time around the track, completing the course, and finding the goal within approximately 39 seconds.

The robot was unable to move towards a ball after identifying it due to a lack of communication between the CMU camera and the navigational system. The robot was also unable to fully navigate the upper level without having some assistance, due to the malfunctioning of the laser range finders at a range closer than 10cm.

Overall, the project was quite successful, and resulted in an autonomous robot that could navigate a course based on a map and feedback from the surrounding objects (ramps, walls, etc). This project provided the opportunity for the students to gain a solid understanding of Mechatronics and the unique difficulties presented when integrating mechanical systems with electronic intelligence. When programming seeks to extend beyond the confines of the processor and initiates I/O communication, there are a myriad of new variables that come into play that would not otherwise be there in the protected and mostly predictable environment that a computer processor provides. Things such as light variance, friction, and weight all present new problems without a simple solution. Each of the students involved in the project gained a deeper appreciation and understanding of the limitations of current computer systems as well as the near-miraculous achievements that technology has been able to perform.

SUGGESTIONS FOR FUTURE IMPROVEMENT

Obvious improvements on the robot's design and implementation center on more effective use of time. Calibrating the laser range-finders, the encoders, the camera, and accurately representing the course with the map file took more time than originally expected. More time would have permitted significantly improved course navigation and allowed the robot to complete each of the outlined tasks with ease.

The robot's overall design was very good but several possible improvements were noted by the team. First, the encoder system lacked accuracy because of several deficiencies in its design. The wheels were "squishy" and may have morphed as they traveled the track, which may have led to inaccurate distance reported by the counting system. Also, the wheels slipped on the surface of the track and led to further inaccuracies. The counting system itself could use a more consistent and permanent black and white (on/off) method rather than using white-out on black tires that get covered in dust and lose their consistency after repeated trials. The laser range-finders had good accuracy when the walls were sufficiently far away from them. At close distances, though, (on the top level) the walls were too close and thus made their readings almost useless. Also, when maneuvering the top level it was noted that the robot's wide berth made it more difficult to get through the narrow passageways. Making the robot narrower would make it easier for it to get through tight squeezes and moving the laser range-finders inside the chassis would allow for a larger distance between them and the course walls.

The robot always had more speed than necessary, and often more speed than was manageable. A gear reduction in the drive motors is recommended. This would not only increase the torque for better ramp climbing capabilities, but it would also relieve the need to change the power input to the motors for each of the different tasks.

The ball catching system worked well at retrieving the balls wherever they were on the course, but it was very ball-specific. In other words, it would be necessary to change the design if

the robot were required to capture balls of varying sizes. Also, because of the nature of the ball catcher retrieving a ball anywhere along the half circle of its hose grabber, it would be important to center the ball after it is captured in order to be able to accurately and consistently “score” when the robot drives straight at the goal and releases the ball into the scoring zone.

Improvements in the project in general might contribute to better robots in the future as well. Simultaneous Internet access and wireless robot access would have helped the team much more effectively do the labs and do research to solve each of the problems encountered while building the robots. If each lab previous to the final project labs was used for implementing what was learned in that lab on the final robot then the robot might have come together much quicker. Also, a more complete Linux image would have saved the team a lot of time because each time the PC 104 system crashed (reached power levels below threshold) the image had to be recopied and reconfigured with the latest changes. The team essentially had to start from scratch every time the computer system failed. Thus a more current image file would have improved the efficiency of development.

II. APPENDIXIS

POWER BUDGET

Energy Producing (Ah)

Device	Voltage	Capacity (Ah)	Energy Stored (Wh)	% Usable Power	Energy Available
Ni-Cd Battery	9.7	3	29.1	80.00%	23.28

Energy Consuming

Componant	Normal Current Draw (A)	Peak Current Draw (A)	Minimum Current Draw (A)	Voltage	Energy use (W)	On/Off ratio (With Normal Current Draw)	Watts used during run time
Micro MO motor (1)	1.2	2	0	9	10.8	0.95	10.26
Micro MO motor (2)	1.2	2	0	9	10.8	0.95	10.26
Maxon motor	1.5	2.5	0	9	13.5	0.15	2.025
TS-5500	0.5	0.8	0.2	5	2.5	1	2.5
netgear card	1	1.3	0	5	5	0.05	0.25
LCD	0.05	0.05	0.05	5	0.25	1	0.25
CMU cam	0.5	0.5	0.5	5	2.5	1	2.5
Camera's Servo	0.6	0.6	0.008	4.8	2.88	0.8	2.304
Phototransistor (1)	0.01	0.01	0.01	5	0.05	1	0.05
Phototransistor (2)	0.01	0.01	0.01	5	0.05	1	0.05
Hall Effect Sensor (1)	0.01	0.01	0.01	5	0.05	1	0.05
Hall Effect Sensor (2)	0.01	0.01	0.01	5	0.05	1	0.05
Hall Effect Sensor (3)	0.01	0.01	0.01	5	0.05	1	0.05
Laser range finder	0.01	0.01	0.01	5	0.05	1	0.05
Total Watts							30.649
Robot Life (h)							0.76

Notes and Definitions

% Usable Power:

Percentage of the battery's capacity that can be used before the battery voltage drops to brown-out level.

Robot Life:

Time until the battery voltage drops to brown out level.

**Many of the values represented on this sheet were found online at the various companies' websites, but some approximations were used where values could not be found.*

SOURCE CODE

a2d.h

```

/*****\
    Title: a2d.h
    Author: Jeremiah K. Jones

    This is the header file for a2d.
/*****/

/*      Vars      */
extern signed int value_LeftFinder; /* last value of Laser Finder */
extern signed int value_RightFinder; /* last value of Laser Finder */
extern signed int a2d_RightEncoder; /* last value of Right Encoder*/
extern signed int a2d_LeftEncoder; /* last value of Left Encoder */
extern signed int a2d_Infrared; /* last value of Infrared sensor */
extern signed int a2d_HallEffect; /* last value of Hall Effect sensor */

/*      Functions      */
int a2d_init(); /* function that initializes a2d
devices*/
int a2d_close(); /* function that closes a2d devices */
void a2d_read(); /* reads and stores current a2d values */

```

a2d.c

```

/*****\
    Title: a2d.c
    Author: Jeremiah K. Jones

    This is the a2d module. It provides functionality for
    Using the A2D pins on the board.
/*****/
/*      Standard Includes      */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>

/*      Non-standard Includes      */
#include "inc/a2d_driver.h"
#include "inc/a2d.h"
#include "inc/debug.h"

/*      Vars to access last A2D reading      */
signed int value_LeftFinder, value_RightFinder ,a2d_RightEncoder,a2d_LeftEncoder,
value_Infrared, value_HallEffect;

/*      Devices (pins) to use      */
int fd_LeftFinder, fd_RightFinder, fd_RightEncoder, fd_LeftEncoder, fd_Infrared,
fd_HallEffect;

/*
    This function initializes the A2D
    pins that will be used for the robot.
    This function should be called before
    attempting to read any A2D pins.
*/
int a2d_init()
{
    /* DEBUGGING OUTPUT */
    debug("Initializing A2D\n");
}

```

```

unsigned char command=0;

/*
    Open all A2D devices
*/
fd_LeftFinder  =open("/dev/AtoD/0",O_RDWR);
fd_RightEncoder  =open("/dev/AtoD/1",O_RDWR);
fd_LeftEncoder  =open("/dev/AtoD/2",O_RDWR);
fd_RightFinder  =open("/dev/AtoD/3",O_RDWR);
fd_Infrared     =open("/dev/AtoD/4",O_RDWR);
fd_HallEffect   =open("/dev/AtoD/5",O_RDWR);

/*
    Ensure all devices have started
    Return -1 if failed
*/
if(fd_LeftFinder<0)
{
    printf("Failed opening Left Laser Range Finder\n");
    return(-1);
}
if(fd_RightEncoder<0)
{
    printf("Failed opening Right Encoder\n");
    return(-1);
}
if(fd_LeftEncoder<0)
{
    printf("Failed opening Left Encoder\n");
    return(-1);
}
if(fd_RightFinder<0)
{
    printf("Failed opening Right Laser Range Finder\n");
    return(-1);
}
if(fd_Infrared<0)
{
    printf("Failed opening Infrared Sensor\n");
    return(-1);
}
if(fd_RightFinder<0)
{
    printf("Failed opening Hall Effect Sensor\n");
    return(-1);
}

/*
    Configure each device (Unipolar/Range=1)
    This value is specified by the "command"
    value.
*/
command = (A2D_UNIPOLAR | A2D_RANGE1); //Specifies Configuration
ioctl(fd_LeftFinder, TSA2D_CONFIG, command);
ioctl(fd_RightEncoder, TSA2D_CONFIG, command);
ioctl(fd_LeftEncoder, TSA2D_CONFIG, command);
ioctl(fd_RightFinder, TSA2D_CONFIG, command);
ioctl(fd_Infrared, TSA2D_CONFIG, command);
ioctl(fd_HallEffect, TSA2D_CONFIG, command);

/*
    Return Normally
*/
return(0);
}

/*
This function closes down the devices
*/

```

```

int a2d_close()
{
    /* DEBUGGING OUTPUT */
    debug("Shutting down A2D\n");

    close(fd_LeftFinder);
    close(fd_RightEncoder);
    close(fd_LeftEncoder);
    close(fd_RightFinder);
    close(fd_Infrared);
    close(fd_HallEffect);

    return(0);
}

/*
    Reads all values in the A2D and stores the
    value in the global vars for each device
*/
void a2d_read()
{
    char tmp[10];
    write(fd_LeftFinder, "trigger", sizeof("trigger"));
    read(fd_LeftFinder, &value_LeftFinder, sizeof(signed int));

    write(fd_RightFinder, "trigger", sizeof("trigger"));
    read(fd_RightFinder, &value_RightFinder, sizeof(signed int));

    write(fd_RightEncoder, "trigger", sizeof("trigger"));
    read(fd_RightEncoder, &a2d_RightEncoder, sizeof(signed int));

    write(fd_LeftEncoder, "trigger", sizeof("trigger"));
    read(fd_LeftEncoder, &a2d_LeftEncoder, sizeof(signed int));

    write(fd_Infrared, "trigger", sizeof("trigger"));
    read(fd_Infrared, &value_Infrared, sizeof(signed int));

    write(fd_Infrared, "trigger", sizeof("trigger"));
    read(fd_HallEffect, &value_HallEffect, sizeof(signed int));
}

```

args.h

```

/*****\
    Title: map.h
    Author: Jeremiah K. Jones

    This is the header file for a map.
\*****/

/*      Prototypes      */
/* function to parse command-line parameters and set options accordingly */
extern void getOptions(int argc, char *argv[]);

/*      Vars      */
extern int debugging;          /* debugging mode (default = false) */
extern int ramp_mode;         /* 'ramp' mode (default = false) */
extern int ball_mode;        /* 'ball' mode (default = false) */
extern int ball_mode_1;      /* 'ball' mode 1 (default = false) */
extern int ball_mode_2;      /* 'ball' mode 2 (default = false) */
extern int ball_capture;     /* 'ball' capture (default = false) */
extern int ball_release;     /* 'ball' release (default = false) */
extern int man_nav_mode;     /* 'manual navigation mode (default = false) */
extern int upper_deck_mode;  /* upper deck mode (default = false) */
extern char *map_file;       /* map filename */
extern char *program_name;   /* name of the program (for errors) */

```

```
extern char *ball_color;          /* color of the ball to find */
```

args.c

```

\*****\
    Title: args.c
    Author: Jeremiah K. Jones

    This module parses command-line parameters and sets the
    program options accordingly.
\*****/

/*      Standard Includes          */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*      Non-standard Includes     */
#include "inc/args.h"
#include "inc/debug.h"
#include "inc/ball.h"

/* Functions          */
static void usage(void);

/*      Vars          */
extern int debugging;          /* debugging mode (default = false) */
int ramp_mode = 0;            /* 'ramp' mode (default = false)   */
int ball_mode = 0;           /* 'ball' mode (default = false)   */
int ball_mode_1 = 0;        /* 'ball' mode 1 (default = false) */
int ball_mode_2 = 0;        /* 'ball' mode 2 (default = false) */
int ball_capture = 0;       /* 'ball' capture (default = false) */
int ball_release = 0;       /* 'ball' release (default = false) */
int man_nav_mode = 0;       /* manual navigation mode          */
int upper_deck_mode = 0;    /* upper deck mode                 */
char *map_file = "map.txt"; /* map filename                   */
char *program_name;        /* name of the program (for errors) */
char *ball_color = "green"; /* color of the ball to find      */

/*
 * Function to pull out options from command line
 * and set program parameters accordingly
 */
void getOptions(int argc, char *argv[])
{
    /* Save the program name for future use */
    program_name = argv[0];

    /* If no arguments were given, print usage and exit
     * if(argc == 1)
     * {
     *     fprintf(stderr, "\nYou need to tell me to do something!\n");
     *     usage();
     *     exit(8);
     * }*/

    /*
     * loop for each option
     * stop if we run out of arguments
     * or we get an argument without a dash
     */
    while ((argc > 1) && (argv[1][0] == '-'))
    {
        /*
         * argv[1][1] is the actual option character
         */
        switch(argv[1][1])
        {

```

```

/*
 * -d turn on debugging
 */
case 'd':
    debugging = 1;
    break;

/*
 * -m<filename> map filename
 * [0] is the dash
 * [1] is the "m"
 * [2] starts the filename
 */
case 'm':
    map_file = &argv[1][2];
    break;

/*
 * -b<color> ball color to find
 * [0] is the dash
 * [1] is the "b"
 * [2] starts the color
 */
case 'b':
    /*      Ball Identification Mode 1      */
    if(argv[1][2] == '1')
    {
        ball_color = &argv[1][3];
        ball_mode_1 = 1;
    }

    /*      Ball Identification Mode 2      */
    else if(argv[1][2] == '2')
    {
        ball_color = &argv[1][3];
        ball_mode_2 = 1;
    }

    /*      Ball capture mode */
    else if(argv[1][2] == 'c')
        ball_capture = 1;

    /*      Ball release mode */
    else if(argv[1][2] == 'r')
        ball_release = 1;

    /*      Ball Identification and Navigation      */
    else
    {
        ball_color = &argv[1][2];
        ball_mode = 1;
    }

    /* Make sure the specified color is green, red, or blue */
    if(strcmp(ball_color, GREEN) != 0 && strcmp(ball_color, RED)
    != 0 && strcmp(ball_color, BLUE) != 0)
    {
        /* if the color is not valid, print the usage and
exit */
        fprintf(stderr, "\n%s is not a valid color.\nYou must
choose red, green, or blue\n", ball_color);
        usage();
    }
    break;

/*
 * -r go up and down ramp
 */
case 'r':
    ramp_mode = 1;
    break;

```

```

/*
 * -u go up to, around, and back down upper deck
 */
case 'u':
    upper_deck_mode = 1;
    break;

/*
 * -n Manual Navigation Mode
 */
case 'n':
    man_nav_mode = 1;
    break;

/*
 * -h print the usage
 */
case 'h':
    usage();
    break;

/*
 * Default if bad argument is detected
 */
default:
    fprintf(stderr, "\nWhat does %s mean? I don't get it!\n",
argv[1]);
        usage();
    }

/*
 * increment the argument list
 * decrement the count
 */
++argv;
--argc;
}

}

/*
 * Usage - tells the user how to use this program and exit.
 */
static void usage(void)
{
    fprintf(stderr, "Syntax: \n\t%s [options]\n", program_name);
    fprintf(stderr, "Options:\n");
    fprintf(stderr, "\t-m<filename>      Use filename (a map) to navigate the
course\n");
    fprintf(stderr, "\t-b<color>          Find and deliver a ball of specified color
(red, green, blue)\n");
    fprintf(stderr, "\t-b1<color>       Track given color (red, green, blue)\n");
    fprintf(stderr, "\t-b2<color>       Approach ball of specified color when seen\n");
    fprintf(stderr, "\t-bc          Capture a ball in front of robot\n");
    fprintf(stderr, "\t-br          Release into scoring bin\n");
    fprintf(stderr, "\t-r          Go up and back down a ramp (robot must be
placed in front of ramp)\n");
    fprintf(stderr, "\t-u          Upper Deck Mode\n");
    fprintf(stderr, "\t-n          Manual Navigation Mode (for testing)\n");
    fprintf(stderr, "\t-d          Turns on debugging\n");
    fprintf(stderr, "\t-h          Print this usage statement\n");
    exit(8);
}

```

ball.h

```

/*****\
Title: ball.h

```

```

Author: Jeremiah K. Jones

This is the header file for the ball module.
\*****/

/* Defines and const*/
const char GREEN[6] = "green";
const char RED[4] = "red";
const char BLUE[5] = "blue";

/* Prototypes */

/*
 * Structures and other
 */

/* structure for balls */
struct ball
{
    char color[6]; /* color of ball */
};

```

ball_mod_1.h

```

\*****\
Title: ball.h
Author: Jeremiah K. Jones

This is the header file for the ball module.
\*****/

/* Defines and const*/
/* green values */
const int gr_rd = 127;
const int gr_gr = 142;
const int gr_bl = 16;
const int gr_rd_d = 20;
const int gr_gr_d = 28;
const int gr_bl_d = 0;
/* red values */
const int rd_rd = 157;
const int rd_gr = 19;
const int rd_bl = 16;
const int rd_rd_d = 29;
const int rd_gr_d = 8;
const int rd_bl_d = 0;
/* blue values */
const int bl_rd = 51;
const int bl_gr = 111;
const int bl_bl = 142;
const int bl_rd_d = 6;
const int bl_gr_d = 26;
const int bl_bl_d = 28;

/* functions */
void bml_run(); /* runs ball mode 1 code */

```

ball_mode_1.c

```

\*****\
Title: ball_mode_1.c

```

```

Author: Jeremiah K. Jones

This module runs the code for ball mode 1
\*****/

/*      Standard Includes          */
#include <string.h>
#include <stdio.h>

/*      Non-standard Includes     */
#include "inc/ball.h"
#include "inc/ball_mode_1.h"
#include "inc/debug.h"
#include "inc/cmucam.h"

/*      Vars                       */
extern cmucam *biff_cmucam;        /* the camera */
extern char *ball_color;          /* color of the ball to find */

/*
 * Function to identify a ball of the
 * given color
 */
void bml_run()
{
    biff_cmucam = new cmucam("/dev/ttyS1");
    char rMin[3], gMin[3], bMin[3], rMax[3], gMax[3], bMax[3];

    /*      If Red ball          */
    if(strcmp(ball_color, RED)==0)
    {
        debug("Looking for RED\n");
        sprintf(rMin, "%i", rd_rd - rd_rd_d);
        sprintf(rMin, "%i", rd_rd - rd_rd_d);
        sprintf(rMax, "%i", rd_rd + rd_rd_d);

        sprintf(gMin, "%i", rd_gr - rd_gr_d);
        sprintf(gMax, "%i", rd_gr + rd_gr_d);

        sprintf(bMin, "%i", rd_bl - rd_bl_d);
        sprintf(bMax, "%i", rd_bl + rd_bl_d);
    }

    /*      If Green ball      */
    else if(strcmp(ball_color, GREEN)==0)
    {
        debug("Looking for GREEN\n");
        sprintf(rMin, "%i", gr_rd - gr_rd_d);
        sprintf(rMax, "%i", gr_rd + gr_rd_d);

        sprintf(gMin, "%i", gr_gr - gr_gr_d);
        sprintf(gMax, "%i", gr_gr + gr_gr_d);

        sprintf(bMin, "%i", gr_bl - gr_bl_d);
        sprintf(bMax, "%i", gr_bl + gr_bl_d);
    }

    /*      If Blue ball      */
    else if(strcmp(ball_color, BLUE)==0)
    {
        debug("Looking for BLUE\n");
        sprintf(rMin, "%i", bl_rd - bl_rd_d);
        sprintf(rMax, "%i", bl_rd + bl_rd_d);

        sprintf(gMin, "%i", bl_gr - bl_gr_d);
        sprintf(gMax, "%i", bl_gr + bl_gr_d);

        sprintf(bMin, "%i", bl_bl - bl_bl_d);
        sprintf(bMax, "%i", bl_bl + bl_bl_d);
    }
}

```

```

else
{
    debug("No color was found\n");
    exit(8);
}
debug("rMin=");debug(rMin);debug("\n");
debug("gMin=");debug(gMin);debug("\n");
debug("bMin=");debug(bMin);debug("\n");

debug("rMax=");debug(rMax);debug("\n");
debug("gMax=");debug(gMax);debug("\n");
debug("bMax=");debug(bMax);debug("\n");

biff_cmucam->tc(rMin, rMax, gMin, gMax, bMin, bMax, true);
}

```

ballcatchermotor.h

```

/*****\
    Title: ballcatchermotor.hpp
    Author: Ben Despain

    This is the header file for the motors file.
*****/

/*      Defines      */
#define BC_FORWARD 1
#define BC_BACKWARD -1

/*      Variables      */
extern signed int bc_mtr_dir;      /* Current direction of motor */
/*      Functions      */
void bc_run();      /*      This function is for manual navigation      */
void bc_motor_init();      /*      This function initializes the motors      */

```

ballcatchermotor.c

```

/*****\
    Title: ballcatchermotor.cpp
    Author: Ben Despain: modified from code by Jeremiah K Jones.

    This is the motors module.  It allows for driving of two
    motors.
*****/

/*      Standard Includes      */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/io.h>
#include <stdlib.h>

/*      Non-standard Includes      */
#include "inc/ballcatchermotor.h"
#include "inc/motors.h"
#include "inc/debug.h"

#define DIO2_OUT 0x7E      /* DIO2 bits 0-7 accessed at 7E hex. */
#define DIO2_CTRL 0x7D      /* DIO2 Control Bit */
#define DIO2_IN 0x7F      /* DIO2 bits 8-11 accessed at 7F hex. */

/*      Global variables      */
extern unsigned char g_outputByte;      // unsigned char used to write to output I/O port

```

```

signed int bc_mtr_dir;          /*      Current direction of motor      */

/*      Local variables      */
const int break_power = 675; /*      Power of breaks int(0-1000)      */
const int release_time = 8; /*      Time to release the ball in seconds      */

/*      functions      */
void bc_motor_init();
void motorStop();
void bc_orBitToByte(int, unsigned char*);
int checkSwitch();

/*
   This function runs the ball catcher in the direction
   specified by bc_mtr_dir;
*/
void bc_run()
{
    int last_speed = motors_speed;
    motors_speed = 50;
    debug("Running ball catcher\n");
    if(bc_mtr_dir == BC_FORWARD)
    {
        while(checkSwitch()==0)
        {
            g_outputByte = 0x00;
            bc_orBitToByte(0, &g_outputByte);
            bc_orBitToByte(1, &g_outputByte);
            outb(g_outputByte,DIO2_OUT);
        }
        motorStop();
    }
    else if(bc_mtr_dir == BC_BACKWARD)
    {
        g_outputByte = 0x00;
        bc_orBitToByte(0, &g_outputByte);
        bc_orBitToByte(2, &g_outputByte);
        outb(g_outputByte,DIO2_OUT);
        sleep(release_time);
        motorStop();
    }
    else
    {
        debug("That is not a valid BC Motor direction!\n");
    }
    motors_speed = last_speed;
}

/*
   This function initializes the motors
*/
void bc_motor_init()
{
    int ioval;          /* for input/output      */
    debug("Initializing Ball Catcher\n");
    // Set and Check permissions for all Digital I/O
    if(ioperm(0x7A,6,1)<0)
    {
        printf("error accessing Digital I/O");
        exit(8);
    }

    // Set pins 0to3 to inputs of DIO Header 2
    ioval = inb(DIO2_CTRL);          // This reads the current configuration of Header
2
    ioval = ioval & 0xEF;
    ioval = ioval | 0x23;
    outb(ioval,DIO2_CTRL);          /* Send the new configuration to Header 2

```

```

        /*      Clear output bit and reset output      */
        g_outputByte = 0x00;
        outb(g_outputByte, DIO2_OUT);
    }

    /*
        This function stops all motors
    */
    void motorStop()
    {
        /*
            Go backwards
        */
        for(int i=0; i < break_power; i++)
        {
            g_outputByte = 0x00;
            bc_orBitToByte(0, &g_outputByte);
            if(bc_mtr_dir == BC_FORWARD)
                bc_orBitToByte(1, &g_outputByte);
            else if(bc_mtr_dir == BC_BACKWARD)
                bc_orBitToByte(2, &g_outputByte);
            outb(g_outputByte, DIO2_OUT);
        }

        /*
            Turn off motor
        */
        g_outputByte = 0x00;
        outb(g_outputByte, DIO2_OUT);
    }

    /*
        This function checks the ball catcher switch to
        see if it has been triggered
    */
    int checkSwitch()
    {
        int switch_val;
        switch_val = inb(DIO2_IN);
        switch_val = (switch_val & 1);          /*      Strip off bit 1      */
        return(switch_val);
    }

    /*
        Changes bit to Byte through Or (from MechIO)
    */
    void bc_orBitToByte(int bitNum, unsigned char* byte)
    {
        switch(bitNum)
        {
            case 0:
                *byte = *byte | 0x01;
                break;

            case 1:
                *byte = *byte | 0x02;
                break;

            case 2:
                *byte = *byte | 0x04;
                break;

            case 3:
                *byte = *byte | 0x08;
                break;

            case 4:

```

```

        *byte = *byte | 0x10;
        break;

    case 5:
        *byte = *byte | 0x20;
        break;

    case 6:
        *byte = *byte | 0x40;
        break;

    case 7:
        *byte = *byte | 0x80;
        break;

    default:
        break;
    }

    return;
}

```

cmucam.h

```

/*
 * cmucam.h
 * This file facilitates use of the CMU camera
 * Author:      Christopher Wilkinson
 * Class:      IT-548 Mechatronics
 * Data:      December - 2003
 *
 * Still need to implement a flush buffer command.
 */

#ifndef cmucam_h
#define cmucam_h

#include <termios.h>

#include <fcntl.h>
#include <unistd.h>

#include <string>
#include <iostream>
#include <sstream>
#include <stdlib.h>

using namespace std;

class cmucam {
    //Friends
        //Friend classes

        //Friend methods

    public:
        //Public variables
            //No public variables

        //Public methods
            cmucam(string port);
            ~cmucam();

            bool GetColor();

        //Public Camera commands
            void init();
            bool idle();
}

```

```

        unsigned int il();
        bool gm(bool multiline);
        bool gv();
        bool rs();
        unsigned int sl(string value);
        bool tc(bool multiline);
        bool tc(string Rmin, string Rmax, string Gmin, string Gmax, string
Bmin, string Bmax, bool multiline);
        bool tw(bool multiline);
        string GetResponse();
        /* This method is for commands using multiline mode. */
        bool GetStreamPacket();

protected:
    //Protected variables

    //Protected methods

private:
    //Private variables
        int fd;
        struct termios oldtio, newtio;
        string response;
        string ColorArray[3];
        string ColorPacket;

    //Private methods
        bool cr (string reg, string value);
        bool dm (string value);
        bool mm (string value);
        bool nf (string value);
        bool pm (string value);
        bool rm (string value);
        bool sm (string value);
        bool sw ();
        bool sw (string x1, string y1, string x2, string y2);

        bool ExecuteCommand(string command, bool multiline);
        bool FormatResponsePacket(char temp[], int size);
        bool FormatResponseString(char temp[]);
        string NumberToString(int number);
};

#endif

```

cmucam.cpp

```

/*
 * cmucam.cpp
 * This file facilitates use of the CMU camera
 * Author:      Christopher Wilkinson
 * Class:      IT-548 Mechatronics
 * Date:      December - 2003
 */

#include "inc/cmucam.h"

#define DEBUG 0

//Constructor
cmucam::cmucam(string port) {

    /* O_RDWR:      Open in read/write mode */
    fd = open (port.c_str(), O_RDWR);
    if(fd == -1) {
        perror(port.c_str());
        exit(-1);
    }
}

```

```

/* Save the current serial port settings */
tcgetattr (fd, &oldtio);

/* Use the same settings of the port changing only what is needed. */
newtio = oldtio;

/* Set the input/output baud rates for this device */
cfsetispeed (&newtio, B115200);
cfsetospeed (&newtio, B115200);

/* CLOCAL:      Local connection (no modem control) */
/* CREAD:       Enable the receiver */
newtio.c_cflag |= (CLOCAL | CREAD);

/* PARENB:      Use NO parity */
/* CSTOPB:      Use 1 stop bit */
/* CSIZE:       Next two constants: */
/* CS8:         Use 8 data bits */
newtio.c_cflag &= ~PARENB;
newtio.c_cflag &= ~CSTOPB;
newtio.c_cflag &= ~CSIZE;
newtio.c_cflag |= CS8;

/* Disable hardware flow control */
newtio.c_cflag &= ~(CRTSCTS);

/* ICANON:      Disable Canonical mode */
/* ECHO:        Disable echoing of input characters */
/* ECHOE:       Echo erase characters as BS-SP-BS */
/* ISIG:        Disable status signals */
newtio.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

/* IGNPAR:      Ignore bytes with parity errors */
/* ICRNL:       Map CR to NL (otherwise a CR input on the other computer will not
terminate input) */
newtio.c_iflag |= (IGNPAR | ICRNL);

/* IXON:        Disable software flow control (incoming) */
/* IXOFF:       Disable software flow control (outgoing) */
/* IXANY:       Disable software flow control (any character can start flow control
*/
newtio.c_iflag &= ~(IXON | IXOFF | IXANY);

/* Raw output */
newtio.c_oflag &= ~OPOST;

/* Clean the modem line and activate new port settings */
tcflush(fd, TCIFLUSH);
tcsetattr(fd, TCSANOW, &newtio);

/* Initialize the camera */
init();
}

/* Destructor - Reset the camera, serial port settings, and then close it. */
cmucam::~cmucam () {
    rs();
    tcsetattr(fd, TCSANOW, &oldtio);
    close(fd);
}

void cmucam::init() {
    rs ();
    mm ("2");
}

bool cmucam::GetColor () {
    gm (false);
    ColorPacket = response;
    cout << "Color packet = " << ColorPacket << "\n";
}

```

```

    }

bool cmucam::cr (string reg, string value) {return ExecuteCommand ("cr " + reg + " " +
value, false);}

bool cmucam::dm (string value) {return ExecuteCommand ("dm " + value, false);}

bool cmucam::gv () {return ExecuteCommand ("gv", false);}

bool cmucam::idle() {return ExecuteCommand ("", false);}

bool cmucam::mm (string value) {return ExecuteCommand ("mm " + value, false);}

bool cmucam::nf (string value) {return ExecuteCommand ("nf " + value, false);}

bool cmucam::pm (string value) {return ExecuteCommand ("pm " + value, false);}

bool cmucam::rm (string value) {return ExecuteCommand ("rm " + value, false);}

bool cmucam::rs () {return ExecuteCommand ("rs", false);}

bool cmucam::sm (string value) {return ExecuteCommand ("sm " + value, false);}

bool cmucam::sw () {return ExecuteCommand ("sw", false);}

bool cmucam::gm (bool multiline) {
    bool result;
    if (multiline) {
        bool result = true;
    }
    else {
        pm ("1");
        bool result = false;
    }
    rm ("3");
    string command = "gm\r";
    int written = write(fd, command.c_str(), command.length());
    if(written > 0) {
        if (!multiline) {
            char temp[9];
            int readout = read(fd, temp, 9);
            result = FormatResponsePacket(temp, 9);
            string temp_response = response;
            rm ("0");
            pm ("0");
            response = temp_response;
        }
    }
    else {
        perror("There was an error writing to the port...\n");
    }
    return result;
}

bool cmucam::tc (bool multiline) {
    bool result;
    if (multiline) {
        result = true;
    }
    else {
        pm ("1");
        result = false;
    }
    rm ("3");
    string command = "tc\r";
    int written = write(fd, command.c_str(), command.length());
    if(written > 0) {
        if (!multiline) {
            char temp[11];
            int readout = read(fd, temp, 11);
            result = FormatResponsePacket(temp, 11);
        }
    }
}

```

```

        string temp_response = response;
        pm ("0");
        rm ("0");
        response = temp_response;
    }
}
else {
    perror("There was a problem writing to the port...\n");
}
return result;
}

bool cmucam::tc (string Rmin, string Rmax, string Gmin, string Gmax, string Bmin, string
Bmax, bool multiline) {
    bool result;
    if (multiline) {
        result = true;
    }
    else {
        pm ("1");
        result = false;
    }
    rm ("3");
    string command = "tc ";
    command += Rmin + " " + Rmax + " " + Gmin + " " + Gmax + " " + Bmin + " " + Bmax +
"\r";
    int written = write(fd, command.c_str(), command.length());
    if(written > 0) {
        if (!multiline) {
            char temp[11];
            int readout = read(fd, temp, 11);
            result = FormatResponsePacket(temp, 11);
            string temp_response = response;
            rm ("0");
            pm ("0");
            response = temp_response;
        }
    }
    else {
        perror("There was an error writing to the port...\n");
    }
    return result;
}

bool cmucam::tw (bool multiline) {
    bool result;
    if (multiline) {
        result = true;
    }
    else {
        pm ("1");
        result = false;
    }
    rm ("3");
    string command = "tw\r";
    int written = write(fd, command.c_str(), command.length());
    if(written > 0) {
        if (!multiline) {
            char temp1[9];
            int readout = read(fd, temp1, 9);
            char temp2[11];
            readout = read(fd, temp2, 11);
            result = FormatResponsePacket(temp2, 11);
            string temp_response = response;
            rm ("0");
            pm ("0");
            response = temp_response;
        }
    }
    else {
        perror("There was an error writing to the port...\n");
    }
}

```

```

    }
    return result;
}

bool cmucam::FormatResponsePacket (char temp[], int size) {
    response = temp[1];
    for (int i = 2; i < size - 1; i++) {
        string number;
        if ((int)temp[i] < 0) {
            number = NumberToString ((int)temp[i] + 256);
        }
        else {
            number = NumberToString ((int)temp[i]);
        }
        response = response + " " + number;
    }
    return true;
}

string cmucam::NumberToString (int number) {
    std::ostringstream number_str;
    number_str << number;
    return number_str.str();
}

/* Execute one line returning commands and return results */
/* If multiline output, enter true for multiline. */
bool cmucam::ExecuteCommand(string command, bool multiline){
    bool result = false;
    command += "\r";
    int written = write(fd, command.c_str(), command.length());
    if (written > 0) {
        char temp[80];
        int readout = read (fd, temp, 80);
        temp[readout] = 0;
        result = FormatResponseString(temp);
    }
    else {
        perror("There was an error writing to the port...\n");
    }
    return result;
}

/* Provide a protected way to retrieve the response */
string cmucam::GetResponse() {
    return response;
}

/* Remove prefix/suffix info from response */
bool cmucam::FormatResponseString(char temp[]) {
    bool result = false;
    response = temp;
    if (response.substr (0,3) == "ACK") {
        response = response.substr (3);
        result = true;
        while (response.substr (0, 1) == "\n") {
            response = response.substr (1);
        }
        int pos = response.rfind (":");
        if (pos != string::npos) {
            response = response.substr (0, pos - 1);
        }
        if (response == ":") {
            response = "SUCCESS";
        }
    }
    else {
        response = "FAILURE";
    }
    //cout << "processed response = '" << response << "'\n";
    return result;
}

```

```
}

```

debug.h

```

/*****\
    Title: debug.h
    Author: Jeremiah K. Jones

    This is the header file for the debugging module.
\*****/

/*    Defines          */
/*    Prototypes      */

/* prints a simple string */
void debug(char*);

/*    Vars          */
extern int debugging; /* will be true if in debugging mode */

```

debug.c

```

/*****\
    Title: debug.c
    Author: Jeremiah K. Jones

    This is the debugging module. It allows for quick print
    statements to be used when in debugging mode.
\*****/

/*    Standard Includes          */
#include <stdio.h>

/*    Non-standard Includes      */
#include "inc/debug.h"

/*    Vars          */
int debugging = 0; /* indicates wether debugging is on */

/* This function prints the given message if debugging is on */
void debug(char *statement)
{
    if(debugging == 1) fprintf(stderr,statement);
}

```

encoders.h

```

/*****\
    Title: encoders.h
    Author: Jeremiah K. Jones

    This is the header file for the encoders.
\*****/

/*    Defines and const*/

/*    Variables          */
extern int l_enc_val; /* Current value of left encoder */
extern int r_enc_val; /* Current value of right encoder */
extern signed int a2d_RightEncoder; /* last value of Right Encoder*/
extern signed int a2d_LeftEncoder; /* last value of Left Encoder */

```

```

/*      Functions      */
void enc_left_rst(); /* Resets the left encoder */
void enc_right_rst(); /* Resets the right encoder */
void enc_init(); /* Initialize encoders */
void doEncoders(); /* Increments or Decrements the encoder values */

```

encoders.c

```

\*****\
Title: encoders.c
Author: Jeremiah K. Jones

This is the encoders module. It is for encoders on both
a left and right motor. It increments or decrements the
values of global variables used for tracking the encoders.
\*****/

/*      Standard Includes      */
#include <stdio.h>

/*      Non-standard Includes      */
#include "inc/encoders.h"
#include "inc/motors.h"
#include "inc/a2d.h"
#include "inc/debug.h"

/*      Global Variables      */
extern signed int a2d_RightEncoder; /* last value of Right Encoder*/
extern signed int a2d_LeftEncoder; /* last value of Left Encoder */
int l_enc_val = 0; /* Current value of left encoder */
int r_enc_val = 0; /* Current value of right encoder */

/*      Local Variables and defines      */
#define WHITE 1
#define BLACK 0
int l_enc_color;
int r_enc_color;
const float ENC_THRESH = 2000; /* Threshold of the encoders */

/*      Functions      */
void enc_init(); /* Initializes the encoders */
void enc_left_rst(); /* Resets the left encoder */
void enc_right_rst(); /* Resets the right encoder */
int encGetLeftColor(); /* Returns the color of the left encoder */
int encGetRightColor(); /* Returns the color of the right encoder */
void doEncoders(); /* Increments or Decrements global encoder
values */

/*
Initialize the encoders
*/
void enc_init()
{
    enc_left_rst();
    enc_right_rst();
}

/*
Resets the left encoder
*/
void enc_left_rst()
{
    a2d_read();

```

```

        l_enc_color = encGetLeftColor();
        l_enc_val = 0;
    }

    /*
     * Resets the right encoder
     */
    void enc_right_rst()
    {
        a2d_read();
        r_enc_color = encGetRightColor();
        r_enc_val = 0;
    }

    /*
     * Returns the color of the left encoder
     */
    int encGetLeftColor()
    {
        if(a2d_LeftEncoder < ENC_THRESH)
            return WHITE;
        return BLACK;
    }

    /*
     * Returns the color of the right encoder
     */
    int encGetRightColor()
    {
        if(a2d_RightEncoder < ENC_THRESH)
            return WHITE;
        return BLACK;
    }

    /*
     * Checks the encoders and changes global values
     * based on current output of encoders
     */
    void doEncoders()
    {
        /*
         * Remember previous values
         */
        int old_left = l_enc_color;
        int old_right = r_enc_color;

        /*
         * Read the a2d
         */
        a2d_read();

        /*
         * Convert to colors
         */
        l_enc_color = encGetLeftColor();
        r_enc_color = encGetRightColor();

        /*
         * If the value has changed, then inc or
         * dec the global value based on motor direction
         */
        if(l_enc_color != old_left)
            l_enc_val++;

        if(r_enc_color != old_right)
            r_enc_val++;
    }
}

```

infrared.h

```

/*****\
    Title: infrared.h
    Author: Jeremiah K. Jones

    This is the header file for the infrared finder used to
    locate the goal at the end of the track.
\*****/

/*    Defines and const*/

/*    Variables          */
extern int value_Infrared; /*    Value of the infrared */
extern int motors_speed; /*    Speed of motors          */
extern int brake_power; /*    Power of brakes          */

/*    Functions          */
int psFind(); /*    Searches for the finder and zeros in on it */

```

infrared.c

```

/*****\
    Title: infrared.c
    Author: Jeremiah K. Jones

    This is the infrared module. It is used to detect and zero
    in on the goal at the end of the track.
\*****/

/*    Standard Includes          */
#include <stdio.h>

/*    Non-standard Includes      */
#include "inc/infrared.h"
#include "inc/motors.h"
#include "inc/a2d.h"
#include "inc/debug.h"

/*    Global Variables          */
#define INF_THRESH 575 /*    Indicates we are close to the goal */
extern signed int value_Infrared; /*    last value of Infrared*/

/*    Local Variables          and defines          */
int ps_currVal = 0; /*    Holds the current finder reading */
int ps_lastVal = 0; /*    Holds the previous finder reading */
int ps_offset = 50; /*    Offset for changing lf resolutions */
extern int last_speed; /*    To save the last motor speed */
extern int last_brake; /*    To save the last breaking power */

/*    Functions          */
int psFind(); /*    Moves the robot towards the goal */
int ps_goToDec(); /*    Moves motor until reaches decreasing value */
void psReset(); /*    Resets the finder values */
void psSwitchDir(); /*    Changes the motor's direction */

/*

```

```

        Returns the value of the left finder
*/
int psFind()
{
    last_speed = motors_speed;
    motors_speed = 50;
    last_brake = brake_power;
    brake_power = 0;

    debug("Trying to locate infrared lights\n");
    psReset();
    a2d_read();          /*      Get first reading      */
    ps_lastVal = value_Infrared; /*      Store reading      */
    ps_lastVal = ps_lastVal + ps_offset;
    mtr_left_dir = MOTORS_FORWARD; /*      Set initial direction */
    mtr_right_dir = MOTORS_FORWARD; /*      Set initial direction */
    rightMotorGo(2); /*      Move one increment      */

    a2d_read();          /*      Get reading      */

    ps_currVal = value_Infrared;
    if(ps_currVal < INF_THRESH)
        return(1);
    if(ps_currVal < ps_lastVal)
    {
        if(ps_goToDec()==1) return(1); /* Go until value decreases */
        motors_speed = last_speed;
        brake_power = last_brake;
    }
    else if(ps_currVal > ps_lastVal)
    {
        psSwitchDir(); /*      Reverse motors */
        if(ps_goToDec()==1) return(1); /*      Go until value decreases */
    }
    /*
        motors_speed = last_speed;
        brake_power = last_brake;
    */
    }
    else
    {
        debug("Error while locating infrared array\n");
        motors_speed = last_speed;
        brake_power = last_brake;
    }
    return(0);
}

/*
    This function moves the motor until the finder's
    value begins to decrease
*/
int ps_goToDec()
{
    debug("Trying to find min value\n");
    int max_tries = 100; /*      Maximum number of times to try and center      */
    for(int i=0; i < max_tries; i++)
    {
        rightMotorGo(1); /* Move one increment */

        a2d_read();          /*      Get Reading      */
        ps_lastVal = ps_currVal + ps_offset; /*      Store last value      */
        ps_currVal = value_Infrared;
        if(ps_currVal < INF_THRESH)
            return(1);
        if(ps_currVal > ps_lastVal)
        {
            psSwitchDir();
            rightMotorGo(2); /* Move one increment */
            motors_speed = last_speed;
            brake_power = last_brake;
        }
    }
}

```

```

        return(0);
    }
}
debug("Exceeded Maximum Tries\n");
psReset();
motors_speed = last_speed;
brake_power = last_brake;
return(0);
}

/*
   This function resets the variables that hold
   the lf values
*/
void psReset()
{
    ps_currVal = 0;          /* Holds the current finder reading */
    ps_lastVal = 0;        /* Holds the previous finder reading */
}

/*
   This function changes the motor directions
*/
void psSwitchDir()
{
    debug("Swapping motor direction\n");
    /* Swap left */
    if(mtr_left_dir == MOTORS_FORWARD)
        mtr_left_dir = MOTORS_BACKWARD;
    else mtr_left_dir = MOTORS_FORWARD;

    /* Swap right */
    if(mtr_right_dir == MOTORS_FORWARD)
        mtr_right_dir = MOTORS_BACKWARD;
    else mtr_right_dir = MOTORS_FORWARD;
}

```

laser_finders.h

```

\*****\
Title: laser_finders.h
Author: Jeremiah K. Jones

This is the header file for the laser finders. This module
is used to re-center the robot after making a right or left
90 degree turn. This can only be used if there is a wall
present, or some other flat object for the signal to be
bounced off of.
\*****/

#include "map.h"

/* Defines and const*/

/* Variables */
extern int motors_speed; /* Speed of motors */
extern int brake_power; /* Power of brakes */

/* Functions */
void lfCenter(enum directions); /* Centers the robot based based on the given
direction */

```

laser_finders.c

```

/*****\
  Title: laser_finders.c
  Author: Jeremiah K. Jones

  This is the laser finders module.  It uses laser range
  finders on the left and right side of the robot to re-center
  the robot after turning.  This will only work when near a
  wall or other device that the signal can be reflected off
  of.
*****/

/*      Standard Includes          */
#include <stdio.h>

/*      Non-standard Includes      */
#include "inc/laser_finders.h"
#include "inc/motors.h"
#include "inc/a2d.h"
#include "inc/debug.h"

/*      Global Variables           */
extern signed int a2d_RightFinder; /* last value of Right Finder*/
extern signed int a2d_LeftFinder;  /* last value of Left Finder */

/*      Local Variables            and defines          */
int currVal = 0; /* Holds the current finder reading */
int lastVal = 0; /* Holds the previous finder reading */
enum directions whichFinder; /* Specifies left or right finder */
int lf_offset = 8; /* Offset for changing lf resolutions */
int last_speed; /* To save the last motor speed */
int last_brake; /* To save the last breaking power */

/*      Functions                  */
void lfCenter(enum directions); /* Centers the robot based based
on the given direction */
void goToDec(); /* Moves motor until reaches decreasing value */
void lfReset(); /* Resets the finder values */
void switchDirection(); /* Changes the motor's direction */

/*
  Returns the value of the left finder
*/
void lfCenter(enum directions dir)
{
    last_speed = motors_speed;
    motors_speed = 30;
    last_brake = brake_power;
    brake_power = 0;
    debug("Trying to center Biff\n");
    lfReset();
    whichFinder = dir;
    a2d_read(); /* Get first reading */
    if(whichFinder == EAST)
        lastVal = value_LeftFinder; /* Store reading */
    else lastVal = value_RightFinder;
    lastVal = lastVal - lf_offset;
    mtr_left_dir = MOTORS_FORWARD; /* Set initial direction */
    mtr_right_dir = MOTORS_FORWARD; /* Set initial direction */
    if(whichFinder == EAST)
        leftMotorGo(1); /* Move one increment */
    else rightMotorGo(1); /* Move one increment */

    a2d_read(); /* Get reading */

    if(whichFinder == EAST)

```

```

        currVal = value_LeftFinder; /* Save reading*/
    else currVal = value_RightFinder;

    if(currVal > lastVal)
    {
        goToDec(); /* Go until value decreases */
        motors_speed = last_speed;
        brake_power = last_brake;
        return;
    }
    if(currVal < lastVal)
    {
        switchDirection(); /* Reverse motors */
        goToDec(); /* Go until value decreases */
        motors_speed = last_speed;
        brake_power = last_brake;
        return;
    }
    else
    {
        debug("Error while centering\n");
        motors_speed = last_speed;
        brake_power = last_brake;
        return;
    }
}

/*
   This function moves the motor until the finder's
   value begins to increase
*/
void goToDec()
{
    debug("Trying to find max value\n");
    int max_tries = 100; /* Maximum number of times to try and center */
    for(int i=0; i < max_tries; i++)
    {
        if(whichFinder == EAST)
            leftMotorGo(1); /* Move one increment */
        else rightMotorGo(1); /* Move one increment */

        a2d_read(); /* Get Reading */
        lastVal = currVal - lf_offset; /* Store last value */
        if(whichFinder == EAST)
            currVal = value_LeftFinder; /* Save reading*/
        else currVal = value_RightFinder;
        if(currVal < lastVal)
        {
            switchDirection();
            if(whichFinder == EAST)
                leftMotorGo(1); /* Move one increment */
            else rightMotorGo(1); /* Move one increment */
            motors_speed = last_speed;
            brake_power = last_brake;
            return;
        }
    }
    debug("Exceeded Maximum Tries\n");
    lfReset();
    motors_speed = last_speed;
    brake_power = last_brake;
    return;
}

/*
   This function resets the variables that hold
   the lf values
*/

```

```

void lfReset()
{
    currVal = 0; /* Holds the current finder reading */
    lastVal = 0; /* Holds the previous finder reading */
}

/*
   This function changes the direction of the
   motors
*/
void switchDirection()
{
    debug("Swapping motor direction\n");
    /* Swap left */
    if(mtr_left_dir == MOTORS_FORWARD)
        mtr_left_dir = MOTORS_BACKWARD;
    else mtr_left_dir = MOTORS_FORWARD;

    /* Swap right */
    if(mtr_right_dir == MOTORS_FORWARD)
        mtr_right_dir = MOTORS_BACKWARD;
    else mtr_right_dir = MOTORS_FORWARD;
}

```

map.h

```

/*****\
    Title: map.h
    Author: Jeremiah K. Jones

    This is the header file for loading a map.
/*****/

/*
 * This defines the various types of objects/locations that
 * the robot may encounter.
 */
enum location_types {PATH, OPEN, RAMP, DIP, WALL, HOME, START, TRAP};

/*
 * This defines the different directions used for navigation
 */
enum directions {NORTH, EAST, SOUTH, WEST, NEAST, NWEST, SEAST, SWEST};

/*
 * This represents a position.
 */
struct position
{
    int id; /* numerical
identifier */
    enum directions nxt_path_dir; /* direction of next path */
    enum location_types type; /* the type of location */
    enum location_types nxt_dest; /* next destination type */
    enum location_types n_type; /* type to the north */
    enum location_types e_type; /* type to the east */
    enum location_types s_type; /* type to the south */
    enum location_types w_type; /* type to the west */
    int n_dist; /* distance to north */
    int e_dist; /* distance to east */
    int s_dist; /* distance to south */
    int w_dist; /* distance to west */
}

```

```

};

/*
 * This is the maximum number of positions, or lines that
 * a map may have. It cannot be a const, because it gives
 * a compile error when trying to declare the size of an
 * array.
 */
#define MAX_POSITIONS 50

/*
 * This is where the map will be stored.
 * It is simply an array of positions, ordered
 * by their "id". The map cannot contain more than
 * the defined number of elements.
 */
extern struct position map_array[];

/*
 * This is where the current position should be stored
 */
extern struct position curr_pos;

/*
 * Vars
 */
extern int debugging; /* debugging mode (default = false) */
extern int ramp_mode; /* 'ramp' mode (default = false) */
extern int ball_mode; /* 'ball' mode (default = false) */
extern char *map_file; /* map filename
 */
extern char *ball_color; /* color of the ball to find
 */

/*
 * Functions
 */
int loadNextPos(); /* loads next position into current
position*/
void loadMap(); /* function that loads the map
 */

```

main.c

```

/*****\
Title: main.c
Author: Jeremiah K. Jones

This is the main file for running the mechatronics robot.
Everything starts from here!

Options:
    [
        -b[color]
        -b1
        -b2[color]
        -bc
        -br
        -u
        -m[map]
        -r
        -d
        -h
    ]

If the -b option is specified, then the robot enters
"ball-seeking" mode. If no color is specified, the
default is "green".

If the -m option is specified, then the robot enters
"map following" mode. If no map is specified, then

```

```

the default is "map.txt"

If the -r option is specified, then the robot enters
"ramp-climbing" mode.

-d turns debugging on. This must go before any -b,-m, or -r.

-h will print the usage message
\*****/

/*      Standard Includes      */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*      Custom Includes      */
#include "inc/debug.h"
#include "inc/args.h"
#include "inc/a2d.h"
#include "inc/motors.h"
#include "inc/servo.h"
#include "inc/cmucam.h"
#include "inc/ball_mode_1.h"
#include "inc/encoders.h"
#include "inc/navigate.h"
#include "inc/ballcatchermotor.h"
#include "inc/laser_finders.h"
#include "inc/infrared.h"

/*
   External Vars
*/
/*      ARGS      */
extern int debugging; /* debugging mode (default = false) */
extern int ramp_mode; /* 'ramp' mode (default = false) */
extern int ball_mode; /* 'ball' mode (default = false) */
extern int ball_mode_1; /* 'ball' mode 1 (default = false) */
extern int ball_mode_2; /* 'ball' mode 2 (default = false) */
extern int ball_capture; /* 'ball' capture mode (default = false) */
extern int ball_release; /* 'ball' release mode (default = false) */
extern int man_nav_mode; /* manual navigation mode mode (default = false) */
extern int upper_deck_mode; /* 'ball' mode (default = false) */
extern char *program_name; /* name of the program (for errors) */
/*      Map      */
extern char *map_file; /* map filename */
extern struct position curr_pos; /* current position */
/*      Ball      */
extern char *ball_color; /* color of the ball to find */
/*      Camera      */
cmucam *biff_cmucam;
/*      Motors      */
extern int motors_speed; /* Speed of motors (1-100) */
extern int brake_power; /* Power of breaks */
/*
   extern int mtr_left_dir; /* Direction of left motor */
   extern int mtr_right_dir; /* Direction of right motor */
*/

/*****
*      Here is the main function!
*      This is where the whole program starts!
*
*      For those who aren't familiar with C syntax, the paramters
*      are:
*          argc (the number of command line arguments, including
*              application name)
*          argv (the arguments)
*****/
int main(int argc, char *argv[])
{

```

```

/* get command-line options and configure program accordingly */
getOptions(argc, argv);

/*
 * Decide what to do based on the options.
 * ie, are we just navigating the track?
 * Or are we going up a ramp?
 * Or are we looking for a ball while going around the track?
 */

/* Are we in ramp mode? */
if(ramp_mode == 1)
{
    /* DEBUGGING OUTPUT */
    debug("Entering ramp mode...\n");

    /* Begin ramp processes */
    if(a2d_init() == -1) return(0);
    motors_init();
    debug("Motors initialized\n");
    run_motor();
    a2d_close();

    /* DEBUGGING OUTPUT */
    debug("Exiting ramp mode...\n");

    /* Exit normally */
    return(0);
}

/* Are we in manual navigation mode? */
if(man_nav_mode == 1)
{
    return(0);
}

/* Are we in upper deck mode? */
if(upper_deck_mode == 1)
{
    debug("Entering upper deck mode...\n");
    return(0);
}

/* Are we in ball mode 1? */
if(ball_mode_1 == 1)
{
    debug("Entering ball identification mode 1...\n");
    bml_run();
    return(0);
}

/* Are we in ball mode 2? */
if(ball_mode_2 == 1)
{
    debug("Entering ball identification mode 2...\n");
    return(0);
}

/* Are we in ball capture mode? */
if(ball_capture == 1)
{
    debug("Entering ball capture mode...\n");
    bc_motor_init();
    bc_mtr_dir = BC_FORWARD;
    bc_run();
    return(0);
}

/* Are we in ball release mode? */
if(ball_release == 1)
{

```

```

    debug("Entering ball release mode...\n");
    if(a2d_init() == -1) return(0);
    motors_init();
    while(psFind() == 0)
    {
        mtr_left_dir = MOTORS_FORWARD;
        mtr_right_dir = MOTORS_FORWARD;
        motors_speed = 50;
        motorsRun(5);
        leftMotorGo(2);
    }
    bc_motor_init();
    bc_mtr_dir = BC_BACKWARD;
    bc_run();
    a2d_close();
    return(0);
}

/*
 * We must be in "map" mode
 */

/* DEBUGGING OUTPUT */
debug("Entering map mode...\nUsing map: ");
debug(map_file);debug("\n");
if(ball_mode == 1)
{
    debug("Looking for a ");
    debug(ball_color);
    debug(" ball\n");
}

/*
 * First we need to find out where we are going
 * by loading the map.
 */
loadMap();

/*
 * Now navigate the course
 */
if(a2d_init() == -1) return(0);
motors_init();
bc_motor_init();
debug("Motors initialized\n");
motors_speed = 50;
navigate();
while(psFind() == 0)
{
    mtr_left_dir = MOTORS_FORWARD;
    mtr_right_dir = MOTORS_FORWARD;
    motors_speed = 50;
    motorsRun(5);
    leftMotorGo(2);
}
a2d_close();

/* DEBUGGING OUTPUT */
debug("Exiting map mode...\n");

/* Exit normally */
/* DEBUGGING OUTPUT */
debug("\nYATTA! SAYONARA!\n");
return(0);
}

```

map.c

```

/*****\

```

Title: map.c
 Author: Jeremiah K. Jones

This is the map module. It defines the way the robot will interpret and handle the path it should take to go around the course.

It should be noted that in this version, the "direction" is always relative to the robot's current position. For example, "North" is always in front of the robot.

The map should be in the following style:

destination	dir	north	distance	east	distance	south
distance	west	distance				

For example:

START	N	WALL	100	WALL	200	RAMP	150	DIP	50
WALL	N	OPEN	0	WALL	100	WALL	75	OPEN	0
WALL	S	WALL	50	WALL	150	HOME	25	OPEN	0
HOME									

```
\*****/
/*      Standard Includes          */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*      Non-standard Includes      */
#include "inc/map.h"
#include "inc/debug.h"

struct position map_array[MAX_POSITIONS]; /* map array */
struct position curr_pos; /* current position */

/*      Functions          */
void matchType(location_types*, char*);

/*
   This is the function that loads the map
*/
void loadMap ()
{
    /* DEBUGGING OUTPUT */
    debug("Loading Map\n");
    debug("Using map: ");
    debug(map_file); debug("\n");

    FILE *in_file; /* The file containing our map */

    /* Let's try to open the file */
    /* DEBUGGING OUTPUT */
    debug("Finding map\n");
    in_file = fopen(map_file, "r");
    if(in_file == NULL)
    {
        fprintf(stderr, "FATAL ERROR: Unable to find the file %s!\n",map_file);
        exit(8);
    }
    /* DEBUGGING OUTPUT */
    debug("Map found!\n");

    char line[200]; /* Used for storing lines from the file */
    int map_ctr = 0; /* Counter for map array */

    /*      Loop to read and process each line of the file */
    /* DEBUGGING OUTPUT */
    debug("Here's the map:\n\n");
    while(fgets(line, sizeof(line), in_file) != NULL)
    {
        /*      temporary variables to store parameters */
```

```

char tmp_d[10] = "", tmp_d_d[2] = "", tmp_n[10] = "", tmp_e[10] = "",
tmp_s[10] = "", tmp_w[10] = "";

/*      temporaty variables to store distances      */
int dist_n = 0, dist_e = 0, dist_s = 0, dist_w = 0;

/*      Grab parameters and store them in temp vars */
sscanf(line, "%s%s%i%s%i%s%i%s%i", tmp_d, tmp_d_d, tmp_n, &dist_n,
tmp_e, &dist_e, tmp_s, &dist_s, tmp_w, &dist_w);
debug(tmp_d); debug("\t"); debug(tmp_n); debug("\t");
debug(tmp_e); debug("\t"); debug(tmp_s); debug("\t");
debug(tmp_w); debug("\n");

/* store this position in the map_array */
struct position tmp_pos;      /* temp position for current pos */
struct position tmp_pos2;     /* temp position for last pos */

tmp_pos.id = map_ctr; /* numerical identifier      */

matchType(&tmp_pos.type, tmp_d);      /* the type of location      */

/* we need to store the destination for the previous position
but we only do this if it is not the first row */
if(map_ctr != 0)
{
    tmp_pos2 = map_array[map_ctr-1];
    matchType(&tmp_pos2.nxt_dest, tmp_d); /* next destination type
*/

    /*      Check for direction of next destination      */
    if(strcmp(tmp_d_d, "N") == 0)
        tmp_pos2.nxt_path_dir = NORTH;
    else if(strcmp(tmp_d_d, "E") == 0)
        tmp_pos2.nxt_path_dir = EAST;
    else if(strcmp(tmp_d_d, "S") == 0)
        tmp_pos2.nxt_path_dir = SOUTH;
    else if(strcmp(tmp_d_d, "W") == 0)
        tmp_pos2.nxt_path_dir = WEST;

    map_array[map_ctr-1] = tmp_pos2;
}

matchType(&tmp_pos.n_type, tmp_n); /* type to the north      */
matchType(&tmp_pos.e_type, tmp_e); /* type to the east      */
matchType(&tmp_pos.s_type, tmp_s); /* type to the south     */
matchType(&tmp_pos.w_type, tmp_w); /* type to the west     */
tmp_pos.n_dist = dist_n;      /* distance to north
*/
tmp_pos.e_dist = dist_e;      /* distance to east
*/
tmp_pos.s_dist = dist_s;      /* distance to south
*/
tmp_pos.w_dist = dist_w;      /* distance to west
*/

/*tmp_pos.nxt_path_dir;      direction of next path      */

/* store it in the array */
map_array[map_ctr] = tmp_pos;

/* increment the array couner */
++map_ctr;
}
/* DEBUGGING OUTPUT */
debug("\nThe map has been analyzed and stored in memory!\n");

/* Now let's close the file */
/* DEBUGGING OUTPUT */
debug("Closing map file\n");
fclose(in_file);

```

```

    /* Set the initial position */
    debug("Initializing map start position\n");
    curr_pos = map_array[0];
}

/*
 * This function will match a string with a location type.
 * It uses pointers as parameters in order to set the values
 * within the map_array.
 */
void matchType(enum location_types *tmp_type, char *tmp_str)
{
    if(strcmp(tmp_str,"START") == 0)
        *tmp_type = START;
    else if(strcmp(tmp_str,"PATH") == 0)
        *tmp_type = PATH;
    else if(strcmp(tmp_str,"OPEN") == 0)
        *tmp_type = OPEN;
    else if(strcmp(tmp_str,"RAMP") == 0)
        *tmp_type = RAMP;
    else if(strcmp(tmp_str,"DIP") == 0)
        *tmp_type = DIP;
    else if(strcmp(tmp_str,"WALL") == 0)
        *tmp_type = WALL;
    else if(strcmp(tmp_str,"HOME") == 0)
        *tmp_type = HOME;
    else if(strcmp(tmp_str,"TRAP") == 0)
        *tmp_type = TRAP;
}

/*
 * This function retrieves the next position and loads
 * it into the curr_pos value. If the next position is
 * home, then it will return 0.
 */
int loadNextPos()
{
    if(curr_pos.nxt_dest != HOME)
    {
        curr_pos = map_array[curr_pos.id+1];
        return(1);
    }
    return(0);
}

```

motors.h

```

/*****\
    Title: motors.hpp
    Author: Jeremiah K. Jones

    This is the header file for the motors file.
\*****/

/* Defines */
#define MOTORS_FORWARD 1
#define MOTORS_BACKWARD -1

/* Variables */
extern signed int mtr_left_dir;          /* Current direction of left motor */
extern signed int mtr_right_dir;        /* Current direction of right motor */
extern int motors_speed;                /* Speed to run the motors at (1-100) */
extern int brake_power;                 /* Power of breaks int(0-1000) */

```

```

/*      Functions      */
void run_motor();          /*      This function is for manual navigation      */
void motors_init();       /*      This function initializes the motors      */
void motorsRight90();     /*      Turns the robot right 90 degrees      */
void motorsLeft90();     /*      Turns the robot left 90 degrees      */
void motorsRun(int dist); /*      Runs the motors a given distance      */
void leftMotorGo(int dist); /*      Runs left motor a given distance      */
void rightMotorGo(int dist); /*      Runs right motor a given distance      */

```

motors.c

```

/*****\
Title: motors.cpp
Author: Jeremiah K. Jones

This is the motors module. It allows for driving of two
motors.
*****/

/*      Standard Includes      */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/io.h>
#include <stdlib.h>

/*      Non-standard Includes      */
#include "inc/debug.h"
#include "inc/a2d_driver.h"
#include "inc/encoders.h"
#include "inc/motors.h"

#define DIO1_R 0x7B /* DIO1 bits 0-7 accessed at 7B hex. */
#define DIO1_L 0x7C /* DIO2 bits 8-10 accessed at 7C hex. */

/*      Global variables      */
unsigned char g_outputByte; // unsigned char used to write to output I/O port
signed int mtr_left_dir;    /*      Current direction of left motor      */
signed int mtr_right_dir;  /*      Current direction of right motor      */
int motors_speed = 100;    /*      Speed to run the motors at      int(1-100)
*/
int brake_power = 500; /*      Power of breaks int(0-1000)      */

/*      Local variables      */
const int r_turn90 = 29; /*      Number of encoder ticks to turn 90 degrees (right
motor)*/
const int l_turn90 = 29; /*      Number of encoder ticks to turn 90 degrees (left
motor)*/
int speed_ctr = 1; /*      Counter for speed (must start at 1)
*/

/*      functions      */
void motors_init();
void rightMotorGo(int);
void leftMotorGo(int);
void motorsStopAll();
void bothMotorsGo(int,int);
void motorsRight90();
void motorsLeft90();
void motorsRun(int);
void orBitToByte(int, unsigned char*);
void speedControl();

/*

```

```

        This function is used in manual navigation.
        It specifies exact functioning of each motor.
*/
void run_motor()
{
    debug("\nGoing up the ramp\n");
    mtr_left_dir = MOTORS_BACKWARD;
    mtr_right_dir = MOTORS_BACKWARD;
    motors_speed = 100;
    motorsRun(250);
}

/*
    This function initializes the motors
*/
void motors_init()
{
    int ioval;          /* for input/output */
    debug("Initializing Motors\n");
    // Set and Check permissions for all Digital I/O
    if(ioperm(0x7A,6,1)<0)
    {
        printf("error accessing Digital I/O");
        exit(8);
    }

    // Set pins 0to3 to inputs of DIO Header 1
    ioval = inb(0x7A);    // This reads the current configuration of Header 1
    ioval = ioval & 0xFF;
    ioval = ioval | 0x23; //Sets DIO1_4-7, DIO1_0-3 & DIO1_8-11 as outputs
    outb(ioval,0x7A);    //Send the new configuration to Header 1

    /*    Clear output bit and reset output    */
    g_outputByte = 0x00;
    outb(g_outputByte, DIO1_R);
    outb(g_outputByte, DIO1_L);
}

/*
    This function moves the left motor. The parameters
    are: distance
*/
void leftMotorGo(int dist)
{
    enc_left_rst();
    while(l_enc_val < dist)
    {
        doEncoders();
        g_outputByte = 0x00;
        orBitToByte(0, &g_outputByte);
        if(mtr_left_dir == MOTORS_FORWARD)
            orBitToByte(2, &g_outputByte);
        else if(mtr_left_dir == MOTORS_BACKWARD)
            orBitToByte(1, &g_outputByte);
        outb(g_outputByte,DIO1_L);
        if(motors_speed != 100)
            speedControl();
    }
    motorsStopAll();
}

/*
    This function moves the right motor. The parameters
    are: distance
*/
void rightMotorGo(int dist)
{
    enc_right_rst();

```

```

while(r_enc_val < dist)
{
    doEncoders();
    g_outputByte = 0x00;
    orBitToByte(0, &g_outputByte);
    if(mtr_right_dir == MOTORS_FORWARD)
        orBitToByte(2, &g_outputByte);
    else if(mtr_right_dir == MOTORS_BACKWARD)
        orBitToByte(1, &g_outputByte);
    outb(g_outputByte, DIO1_R);
    if(motors_speed != 100)
        speedControl();
}
motorsStopAll();
}

/*
   This function stops all motors
*/
void motorsStopAll()
{
    /*
       Go backwards
    */
    for(int i=0; i < brake_power; i++)
    {
        g_outputByte = 0x00;
        orBitToByte(0, &g_outputByte);
        if(mtr_left_dir == MOTORS_FORWARD)
            orBitToByte(1, &g_outputByte);
        else if(mtr_left_dir == MOTORS_BACKWARD)
            orBitToByte(2, &g_outputByte);
        outb(g_outputByte, DIO1_L);

        g_outputByte = 0x00;
        orBitToByte(0, &g_outputByte);
        if(mtr_right_dir == MOTORS_FORWARD)
            orBitToByte(1, &g_outputByte);
        else if(mtr_right_dir == MOTORS_BACKWARD)
            orBitToByte(2, &g_outputByte);
        outb(g_outputByte, DIO1_R);
    }

    /*
       Turn off motors
    */
    g_outputByte = 0x00;
    outb(g_outputByte, DIO1_R);
    outb(g_outputByte, DIO1_L);

    /*
       Re-initialize encoders
    */
    enc_init();
}

/*
   This function moves both motors
*/
void bothMotorsGo(int rdist, int ldist)
{
    enc_init();

    /*
       Get Biggest Distance
    */
    int max_dist = 0;
    if(rdist > ldist) max_dist = rdist;
    else max_dist = ldist;
}

```

```

while(l_enc_val < max_dist || r_enc_val < max_dist)
{
    doEncoders();
    if(l_enc_val < ldist)
    {
        g_outputByte = 0x00;
        orBitToByte(0, &g_outputByte);
        if(mtr_left_dir == MOTORS_FORWARD)
            orBitToByte(2, &g_outputByte);
        else if(mtr_left_dir == MOTORS_BACKWARD)
            orBitToByte(1, &g_outputByte);
        outb(g_outputByte, DIO1_L);
    }
    if(r_enc_val < rdist)
    {
        g_outputByte = 0x00;
        orBitToByte(0, &g_outputByte);
        if(mtr_right_dir == MOTORS_FORWARD)
            orBitToByte(2, &g_outputByte);
        else if(mtr_right_dir == MOTORS_BACKWARD)
            orBitToByte(1, &g_outputByte);
        outb(g_outputByte, DIO1_R);
    }
    if(motors_speed < 100)
        speedControl();
}
motorsStopAll();
}

/*
   This function controls the speed by stopping the
   motors for a specified amount of "time". This is
   not completely accurate due to the fact that it
   relies on the processor cycles and not a real-time
   clock
*/
void speedControl()
{
    int speed_mod = 5;           /* Specifies how often the speed control
takes place */
    if((speed_ctr % speed_mod) == 0)
    {
        int delay = 0;
        int delayMax = 10000;
        int mult = 100;
        if(motors_speed==0) delay=delayMax;
        else if(motors_speed>=100) delay=0;
        else delay = delayMax - (mult*motors_speed);

        if(delay != 0)
        {
            g_outputByte = 0x00;
            outb(g_outputByte, DIO1_R);
            outb(g_outputByte, DIO1_L);
        }

        for(int i=0; i <= delay; i++);
        speed_ctr=1;
    }
    else speed_ctr++;
}

/*
   Changes bit to Byte through Or (from MechIO)
*/
void orBitToByte(int bitNum, unsigned char* byte)
{
    switch(bitNum)
    {

```

```

    case 0:
        *byte = *byte | 0x01;
        break;

    case 1:
        *byte = *byte | 0x02;
        break;

    case 2:
        *byte = *byte | 0x04;
        break;

    case 3:
        *byte = *byte | 0x08;
        break;

    case 4:
        *byte = *byte | 0x10;
        break;

    case 5:
        *byte = *byte | 0x20;
        break;

    case 6:
        *byte = *byte | 0x40;
        break;

    case 7:
        *byte = *byte | 0x80;
        break;

    default:
        break;
}

return;
}

/*
    This function turns the robot right 90 degrees
*/
void motorsRight90()
{
    debug("Turning right 90 degrees\n");
    mtr_right_dir = MOTORS_BACKWARD;
    mtr_left_dir = MOTORS_FORWARD;
    bothMotorsGo(r_turn90, l_turn90);
}

/*
    This function turns the robot left 90 degrees
*/
void motorsLeft90()
{
    debug("Turning left 90 degrees\n");
    mtr_right_dir = MOTORS_FORWARD;
    mtr_left_dir = MOTORS_BACKWARD;
    bothMotorsGo(r_turn90, l_turn90);
}

/*
    This function runs the motors given only a distance
*/
void motorsRun(int dist)
{
    debug("Moving Both Motors\n");
    bothMotorsGo(dist, dist);
}

```

```
}

```

navigate.h

```

/*****\
  Title: navigate.h
  Author: Jeremiah K. Jones

  This is the header file for the navigation module.
\*****/

/*      Functions      */

/*
  This is the main function for navigating through
  the map.  It will use the loaded map array and
  send the appropriate information to the motors to
  navigate through until it reaches "HOME"
*/
void navigate();

```

navigate.c

```

/*****\
  Title: navigate.c
  Author: Jeremiah K. Jones

  This is the navigation module.  It uses a map stored in
  memory to cause the robot to navigate the course.
\*****/

/*      Standard Includes      */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/io.h>
#include <stdlib.h>

/*      Non-standard Includes      */
#include "inc/debug.h"
#include "inc/motors.h"
#include "inc/laser_finders.h"
#include "inc/infrared.h"

/*      Global variables      */
/*
  *      This is where the current position should be stored
  */
extern struct position curr_pos;
/*
  *      This is where the map will be stored.
  *      It is simply an array of positions, ordered
  *      by their "id".  The map cannot contain more than
  *      the defined number of elements.
  */
extern struct position map_array[];

```

```

/*      Functions      */
void navigate();

/*
   This is the main function for navigating through
   the map. It will use the loaded map array and
   send the appropriate information to the motors to
   navigate through until it reaches "HOME"
*/
void navigate()
{
    debug("\n\nStarting to navigate the map\n");
    int runDistance = 0; /*      Distance to run      */
    int prev_speed; /*      Previous speed */

    /*
       Loop through the map until we reach HOME
    */
    do
    {
        struct position next_pos = map_array[curr_pos.id+1];
        if(curr_pos.nxt_path_dir==NORTH)
        {
            mtr_left_dir = MOTORS_FORWARD; /* set direction */
            mtr_right_dir = MOTORS_FORWARD;
            runDistance = next_pos.n_dist; /*      set Distance */
            debug("GOING NORTH\n");
        }
        else if(curr_pos.nxt_path_dir==EAST)
        {
            motorsRight90(); /*      Turn right      */
            lfCenter(EAST); /*      Center the right lf */
            /*
            mtr_left_dir = MOTORS_FORWARD; /* set direction */
            mtr_right_dir = MOTORS_FORWARD;
            runDistance = next_pos.e_dist; /*      set Distance */
            debug("GOING EAST\n");
            */
        }
        else if(curr_pos.nxt_path_dir==SOUTH)
        {
            mtr_left_dir = MOTORS_BACKWARD; /* set direction */
            mtr_right_dir = MOTORS_BACKWARD;
            runDistance = next_pos.s_dist; /*      set Distance */
            debug("GOING SOUTH\n");
        }
        else if(curr_pos.nxt_path_dir==WEST)
        {
            motorsLeft90(); /*      Turn left      */
            lfCenter(WEST); /*      Center the left lf */
            /*
            mtr_left_dir = MOTORS_FORWARD; /* set direction */
            mtr_right_dir = MOTORS_FORWARD;
            runDistance = next_pos.w_dist; /*      set Distance */
            debug("GOING WEST\n");
            */
        }
        else
            return;
        //prev_speed = motors_speed;
        if(curr_pos.type == RAMP)
        {
            debug("DOING RAMP\n");
            motors_speed = 100;
        }
        motorsRun(runDistance);
        //motors_speed = prev_speed;
        motors_speed = 50;
        if(curr_pos.w_type == WALL)
            lfCenter(WEST);
        else if(curr_pos.e_type == WALL)
            lfCenter(EAST);
    }
}

```

```
        }while(loadNextPos());
    }
}
```

servo.h

```

/*****\
    Title: servo.h
    Author: Jeremiah K. Jones

    This is the header file for controlling servos.
\*****/

/*      Vars      */

/*      Functions      */
int servo_init(); /* function that initializes servos*/
int servo_close(); /* function that closes a2d devices */
void SetServo(int iServo, int iPos);/* sets the given servo to the given servo position
*/

```

servo_controller.c

```

/*
* Linux_Servo_FT639 is a Linux servo control program for the FT639 chip.
* Copyright (C) 2001, 2002 Magnus-swe <magnus-swe@telia.com>
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.
*
* Modified by Jeremiah K. Jones November-December 2004
*
*/

#include "inc/servo.h"
#include "inc/debug.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termio.h>
#include <termios.h>

struct termios old_options, new_options;

int device;

/*
    The following originally did not have a type
    the default is int. To avoid warnings, I have
    specified them as int.
*/

```

```

*/
const int setup[]={122};
const int shortpulse[]={85};
const int longpulse[]={90};
const int header[]={96};
const int active[]={117};

unsigned char low_nibble =00001111;           // send this first
unsigned char high_nibble=10000000;          // 0000 0011 1011 1111 for servo 0
char bytes_written[]="";
char serial_port[1024]="";
char control_mode[1024]="";
char center[1024]="";
char input[1024]="";
int keyb_servo_number;
int keyb_servo_position;
int timez=50000;
int result;
int low=0;
int high=255;

int
servo_init()
{
/* Open the serial port /dev/ttyS0 */
debug("\nInitializing servos\n");
device = open (serial_port, O_RDWR | O_NOCTTY | O_NDELAY);
if (device < 0)
{
printf("\nCant open the serial port to initialize servos\n");
exit(1);
}

/* Get the Original port settings */
tcgetattr(device, &old_options);

/* Specify new port settings: BAUDRATE=2400, 8 BITS, No Parity, 1 Stopbit */
new_options.c_cflag = (B2400 | CS8 | CLOCAL); // raw input = no ICANON

/* Specify raw data (deselect some settings) */
new_options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

/* Activate the new port settings */
tcsetattr(device, TCSANOW, &new_options); // TCSAFLUSH

/* Put the chip in setup mode */
write(device, setup, 3);
write(device, shortpulse, 2);
write(device, longpulse, 2);
write(device, header+3, 2);

/* Activate the chip */
write(device, active, 3);
debug("\nDevice Activated.\n");
return(1);
}

int
servo_close()
{
/* Put the chip back in setup mode (for "real" applications) */
debug("\n\nEntering SetupMode and closing serial line\n");

write(device, setup, 4);

/* Put the Original port settings back */
tcsetattr(device, TCSANOW, &old_options); // TCSAFLUSH TCSANOW
close(device);
}

```

```

    return(1);
}

void SetServo( int iServo, int iPos )
{
    unsigned char LowByte, HiByte;
    LowByte = ((iServo-1) << 4) + (iPos&0x0f);
    if (write(device, &LowByte, 1)!=1)
        return;
    HiByte = ((iServo-1) << 4) + (iPos>>4) + (1 << 7);
    if (write(device, &HiByte, 1)!=1)
        return;
}

```

Makefile

```

#This is the Makefile for our robot program
.SUFFIXES: .CPP $(SUFFIXES)

CC      = g++ -march=i386 -mcpu=i386
CCLIBS  = -lm
OSFLAGS_Linux_20 = -DLinux
OSFLAGS_Linux_22 = -DLinux -DLinux_2_2

CCFLAGS      = -g -pipe $(OSFLAGS_Linux_22)
CFLAGS       = $(CCFLAGS)
CXXFLAGS     = $(CCFLAGS)CC=gcc
LDFLAGS      = -L.

# List all .o files
OBJ= src/debug.o src/main.o src/args.o src/a2d.o src/map.o src/motors.o src/cmucam.o
src/ball_mode_1.o src/encoders.o src/navigate.o src/ballcatchermotor.o
src/laser_finders.o src/infrared.o #src/halleffect.o

OUTPUT=goBiff

all: goBiff

goBiff: $(OBJ)
        $(CC) $(CFLAGS) -o $(OUTPUT) $(OBJ)

src/main.o:      src/main.c src/inc/debug.h
src/debug.o:     src/debug.c src/inc/debug.h
src/a2d.o:       src/a2d.c src/inc/a2d.h
src/args.o:      src/args.c src/inc/args.h
src/map.o:      src/map.c src/inc/map.h
src/servo_controller.o:  src/servo_controller.c src/inc/servo.h
src/motors.o:    src/motors.c src/inc/motors.h
src/cmucam.o:    src/cmucam.cpp src/inc/cmucam.h
src/ball_mode_1.o:  src/ball_mode_1.c src/inc/ball_mode_1.h
src/encoders.o:  src/encoders.c src/inc/encoders.h
src/navigate.o:  src/navigate.c src/inc/navigate.h
src/ballcatchermotor.o:  src/ballcatchermotor.c src/inc/ballcatchermotor.h
src/laser_finders.o:  src/laser_finders.c src/inc/laser_finders.h
src/infrared.o:  src/infrared.c src/inc/infrared.h
#src/halleffect.o  src/halleffect.c src/inc/halleffect.h
clean:
    rm -f $(OUTPUT) $(OBJ)

```

Map.txt

```

HOME
START  N      WALL  300   RAMP  0      WALL  300   DIP   200
WALL   E      WALL  0      WALL  315   WALL  800   WALL  100

```


II. BIBLIOGRAPHY

Unless otherwise stated, the documentation is available on the class website.

Class Website (including all datasheets)

<http://www.et.byu.edu/groups/it548>

Fall 2004

Richard Helps and Janell Armstrong

CMUcam Manual v2.00 for the CMUcam v.1.12 firmware

Anthony Rowe and Carnegie Mellon University.

Edited by Charles Rosenberg and Illah Nourbakhsh

Copyright 2003

TS-5500 User's Guide

Technology Systems Inc

January 10, 2003k

Practical C Programming

Steve Oualline

O'Reilly

Copyright 1997

Mechatronics Handbook v1.1

September 2004