PROJECT: ODIN AND WOBBLEHEAD

Origin Detect and Intelligent Navigation Robot and the
Intelligent and Dynamic Charging Station

Michael Apodaca
University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machine Design Laboratory
Dr. Antonio Arroyo

TABLE OF CONTENTS

ABSTRACT	**3**
EXECUTIVE SUMMARY	**4**
INTRODUCTION	**5**
INTEGRATED SYSTEM	**5**
MOBILE PLATFORM	**6**
ACTUATION	**8**
SENSORS	**9**
BEHAVIORS	**11**
CONCLUSION	**12**
APPENDICES	**13**

ABSTRACT

Odin is a six-wheeled robot that is designed to handle climbing of small obstacles and avoidance of large obstacles. WobbleHead is the charging station that allows Odin to charge his batteries when they become low. WobbleHead is a cylindrical shaped and axial actuated charging station. Both WobbleHead and Odin actuate themselves to line up their charge jacks for docking.

EXECUTIVE SUMMARY

The purpose of this project is to design and build a dynamic solution to a charging station. The goal is an actuated charging station that rotates itself to orient its charge jack to point towards a docking robot which also orients itself to line up with the docking station.

Odin is a six-wheeled robot designed around the Mekatronix MRC11 and MRSX01 boards. The robot is designed to handle climbing of small obstacles and avoidance of large obstacles. Infrared Red (IR) sensors will be used to detect which obstacles are passable. Odin's behaviors are to avoid obstacles it finds too large to climb, react to bumping into objects, find the charging station, and dock with the charging station.

WobbleHead is the charging station that allows Odin to charge his batteries when they become low. WobbleHead is a cylindrical shaped and axial actuated charging station designed around the Mekatronix MSCC11 board. WobbleHead detects and tracks Odin using sonar and IR sensors. The behavior of WobbleHead is to orient his charge jack to point towards Odin as Odin navigates around the room and as Odin docks.

Odin was successful in avoiding obstacles while in all states of the motors. Odin was successful in finding the IR emitters of WobbleHead and driving toward them. WobbleHead was successful in finding the location of Odin with sonar. Odin and WobbleHead did not successfully dock. WobbleHead was unable to actuate the motors accurately or use the IR receivers to align with the charge jack of Odin. Odin was not able to make the small adjustments necessary for a successful dock. Ideally, Odin should have the ability to determine the distance to the docking station. This would enable Odin to know to make small adjustments when near WobbleHead. WobbleHead should have more IR receivers to accurately determine actuation at close range. However, the project successfully demonstrated the potential of using a charging station that could dynamically change its orientation to the charging robot.

INTRODUCTION

The purpose of this project is to design and build a dynamic solution to a charging station. The goal is an actuated charging station that rotates itself to orient its charge jack to point towards a docking robot which also orients itself to line up with the docking station.

Odin is a six-wheeled robot designed around the Mekatronix MRC11 and MRSX01 boards. The robot is designed to handle climbing of small obstacles and avoidance of large obstacles. Infrared Red (IR) sensors will be used to detect which obstacles are passable. Odin's behaviors are to avoid obstacles it finds too large to climb, react to bumping into objects, find the charging station, and dock with the charging station.

WobbleHead is the charging station that allows Odin to charge his batteries when they become low. WobbleHead is a cylindrical shaped and axial actuated charging station designed around the Mekatronix MSCC11 board. WobbleHead detects and tracks Odin using sonar and IR sensors. The behavior of WobbleHead is to orient his charge jack to point towards Odin as Odin navigates around the room and as Odin docks.

INTEGRATED SYSTEM

Odin uses the Mekatronix MRC11 and MRSX01 circuit boards. The MRC11 has a Motorola 68HC11 microcontroller with 64 KB of external SRAM. The MRSX01 is a sensor expansion circuit board for the MRC11. The MRSX01 has all the circuitry used to control the 12 IR emitters, and read the 10 IR detectors, 12 bump sensors, battery voltage detector, and charge detector. I had to expand the motor controller circuitry since the MRSX01 only controls two motors. All three motors on each side are controlled by the same signal and therefore turn at the same speed and direction. Odin also includes a sonar transmitter circuit board that is currently not controlled by the microcontroller. However, there is a control pin on the board that can be used to enable or disable the transmitter.

Odin reads all his analog sensors and then converts them to digital values. Odin then determines if he has either bumped into a wall or detected an obstacle with his IR. Odin then arbitrates the motor controls from these algorithms and the state of the battery's charge. If the battery is low then he begins the docking procedure. First, he finds the WobbleHead and points his charge in the direction he found WobbleHead. Second, he begins to follow the IR from WobbleHead until he docks. Otherwise, if the battery is not low, Odin will continue avoid obstacles. Figure 1 shows a block diagram of this system.

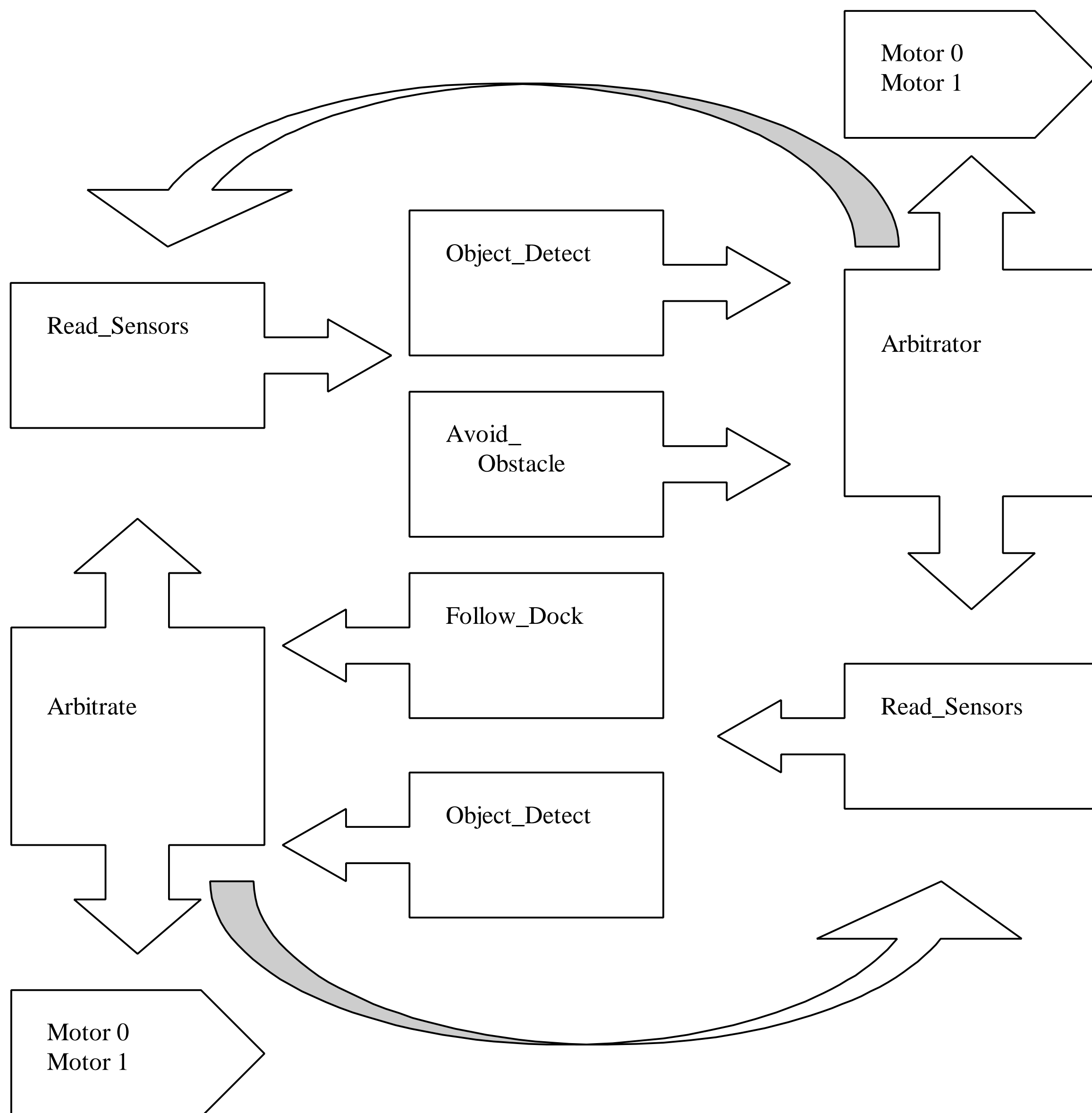


Figure 1: Block Diagram of Odin Software System

WobbleHead uses the Mekatronix MSCC11 circuit board. The MSCC11 has a Motorola HC11 microcontroller with on-board 2KB of EEPROM and 256 bytes of RAM. This board has connectors that allow direct connections to the digital I/O ports, analog ports, and control pins of the processor. WobbleHead has three IR emitters and two IR detectors connected to the MSCC11. He also has three sonar detectors and one motor controller board.

WobbleHead polls the sonar boards until he reads a detection from one of the sonar boards. This determines which board detected the sonar first. WobbleHead then turns its motor if the detection did not originate from the front. Otherwise, he uses his IR detectors to line up with Odin.

MOBILE PLATFORM

Odin is a six-wheeled, double-platform chassis. The platforms are approximately 2 inches apart, while the wheels will be 4 inches high. The large wheels will allow Odin to climb small

obstacles and the six-wheel design should help climbing non-inclined objects. Figure 2 below shows the basic orientation of the chassis.

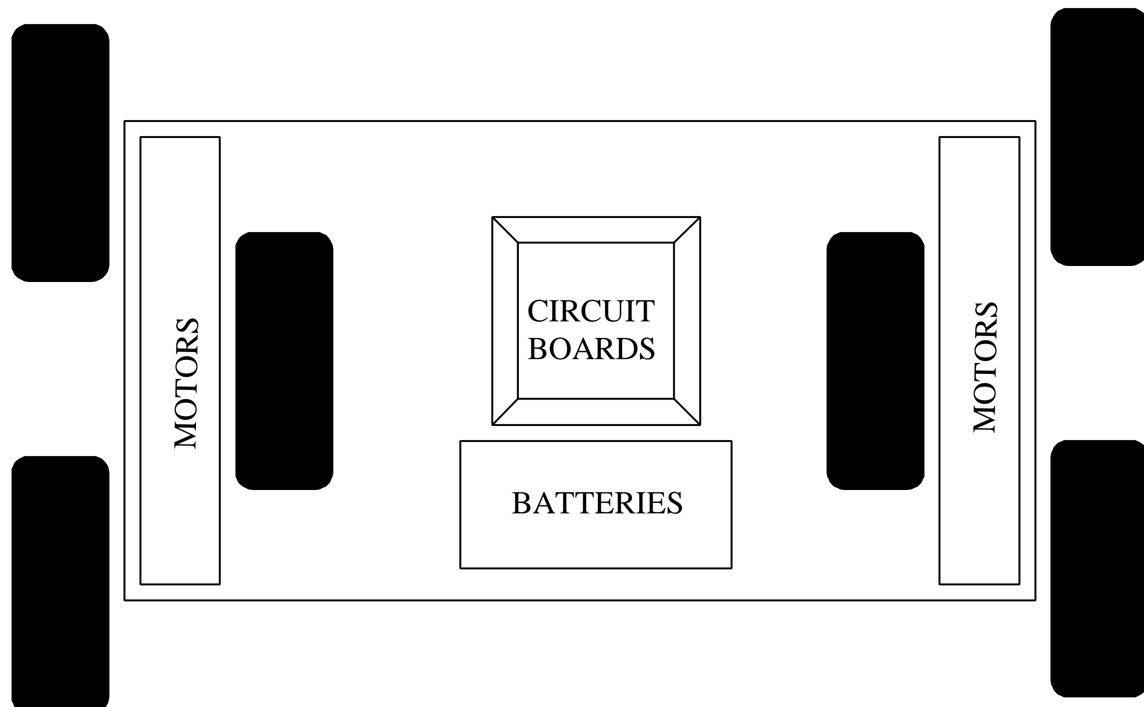


Figure 2: Odin Mobile Platform

The chassis will be rectangular in shape, with the long sides in the front and rear. This will help Odin make sharper turns then square designs. The wheels will also be overlapped to both shorten the overall length of the chassis and support the middle of the chassis from protruding obstacles while climbing. The outer wheels will not be enclosed by the both so they can better climb objects.

Both the front and rear of the body will be the same. This will allow Odin to drive forward and reverse exactly the same. The front panel of the chassis will have the Download/Run, Power On/Power Save, and Reset switches. The rear panel will have the charge jack. This will allow Odin to charge itself in its docking station.

I originally intended the robot to run while upside down. However, the first platform I built was too small for all my circuit boards and batteries to fit inside, so I increased the height of the platform. This made the chassis too tall to run while upside down due solely to the IR emitter mounts. A third design of the upper platform can easily alleviate this problem.

WobbleHead is a cylindrical platform that turns about its vertical axis. The charge jack protrudes from the cylinder horizontally. Figure 3 shows a basic representation of the charging station. WobbleHead can face any direction, thus changing the orientation of its charge jack to the docking robot. The only difficulty with the platform is the size and weight. The small, 3-inch radius I used made it difficult to separate the IR detectors enough to detect from which direction the IR signal of Odin was originating. Also, I needed more weight and surface area on the floor platform to deter WobbleHead from sliding when Odin begins to dock. I have used a rubber mat on the bottom surface to prevent sliding, but there is not enough weight on the lower plate to take advantage of the rubber surface.

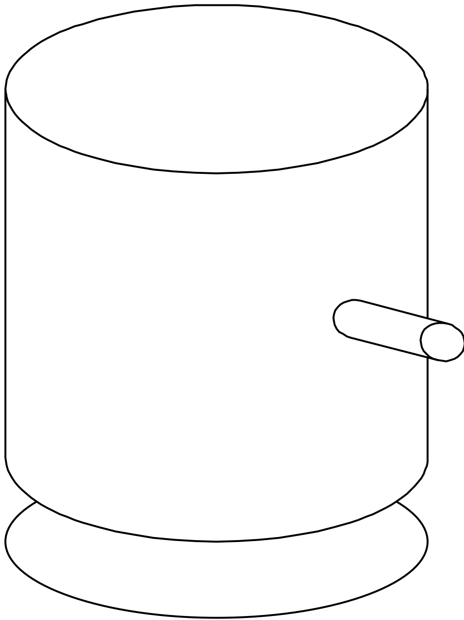


Figure 3: WobbleHead Platform

ACTUATION

Gearhead DC motors will control the six wheels of Odin. The DC motors are actually hacked servomotors used for the gears included in the servo package. There was no need to use high torque motors or high speed motors for this application. The motors used were MS455 dual ball bearing standard servomotors with 42 oz.-in. torque. I also used 4-inch Du-Bro 400RV inflatable wheels. These give Odin shock absorption and greater contact surface with the floor.

The MRSX01 expansion circuit board has only circuitry to control two DC motors. Therefore, I had to expand the circuitry to control all six motors. I removed the L293 Quad Half-H Motor Driver from the MRSX01 and built a circuit board with three L293's in parallel. I then used the control signals from the MRSX01to control the motor drivers. Appendix A1 shows this circuitry.

All six motors are not independent. Bit 6 of Port C (Motor0) from the HC11 controls the direction of the three left motors and bit 7 of Port C (Motor1) controls the right motors. Bit 5 of Port A outputs a pulse-width-modulated signal that controls the speed of the left motors and bit 6 of Port A controls the speed of the right motors. Appendix B3 shows the motor software drivers used for controlling the motors.

A gearhead DC motor will control the spinning of WobbleHead. This motor is also a hacked MS455. A better motor would be a stepping motor since it is much more accurate to control. However, stepper motors do not have enough torque to drive the amount of weight of WobbleHead.

I used the same circuitry for this motor that I used for Odin. The circuit diagram is shown below in Figure 4. The direction control bit is connected to Bit 0 of Port B. The enable control bit is connected to Bit 1 of Port B. The motor driver software is included in "WobbleHead.C" shown in Appendix B2. The motor drivers control the control bits directly. The enable is toggled with 50% pulse-width-modulation at all times, so no speed control is implemented.

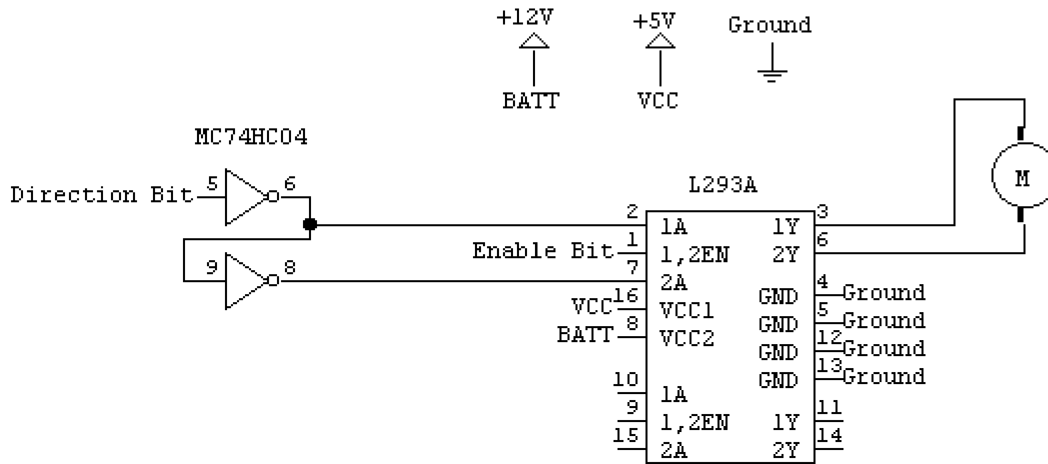


Figure 4: WobbleHead Motor Circuitry

SENSORS

Odin uses 10 Sharp GP1U58Y 40KHz IR Receivers in conjunction with 14 IR emitters modulated at 40KHz to detect objects in front, rear, and on the side of the chassis. The Sharp IR detectors are hacked to output an analog signal using a technique supplied by IMDL. The technique is shown below in Figure 5, copied from the Mekatronix™ Talrik[II] Assemble Manual.

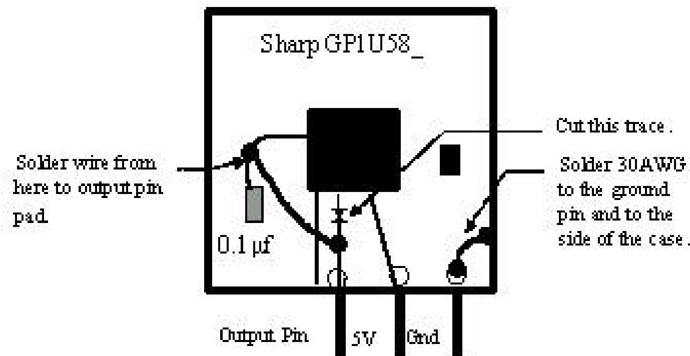


Figure 11 Converting a digital IR sensor to an analog IR sensor requires cutting the trace to the Output pin, soldering the Gnd pin to the side of the case, and connecting the output of the 0.1 µf capacitor to the Output pin.

Figure 5: Analog Hack of Sharp GP1U58Y 40KHz IR Receivers

Odin uses 12 SWPBMT100 Momentary Tactile Switches as bump sensors on both the top and bottom plates to detect objects that are too high for the chassis to clear, objects the wheels did not climb over, and objects not detected by the IR sensors.

Odin also uses a 40KHz Sonar Transmitter for transmitting sonar to WobbleHead. The circuit for the sonar transmitter is shown below in Figure 6. This circuit was able to send a signal approximately 25 feet through air. This was ideal for long range detection of Odin by the sonar receivers on WobbleHead. A circuit diagram in Appendix A2 shows a circuit for generating the 40KHz signal from a 4 MHz crystal oscillator.

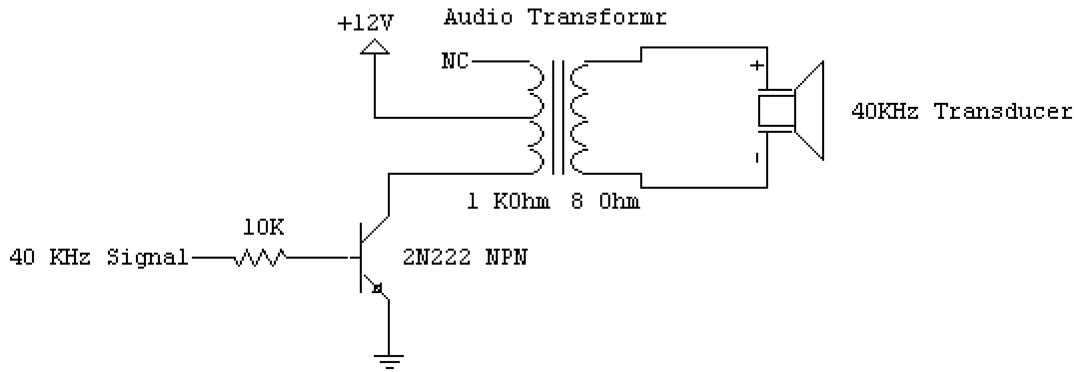


Figure 6: Sonar Transmitter Circuit

WobbleHead also uses three sonar receivers to detect the location of Odin at long range. The circuit for the sonar receiver is shown below in Figure 7. The output of this circuit is an active low pulse-width-modulated signal, where the length of the low pulse corresponds to the length between the receiver and transmitter.

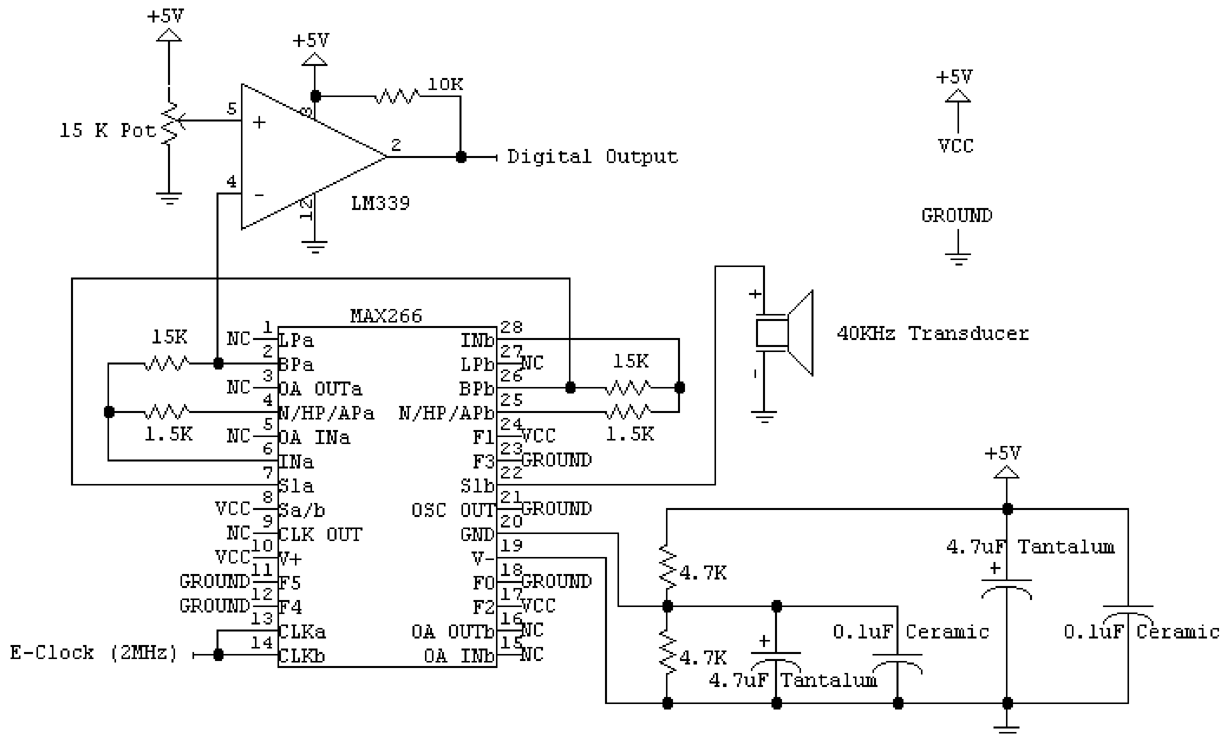


Figure 7: Sonar Receiver Circuit

WobbleHead uses two hacked Sharp IR receivers and three IR emitters to align with Odin at close range. The IR receivers detect the IR emitters of Odin and the IR emitters are used so the IR receivers of Odin can detect WobbleHead.

I was supplied the circuits for the sonar from a design used by IMDL. However, the circuit I received had several errors, which I corrected. First, the circuit I used included the circuitry for generating the 40KHz signal into the sonar transmitter, shown in Appendix A2. The 74HC74 D flip-flop was wired incorrectly. The D input, pin 2, was wired to the Q output, pin 5, and VCC. Instead, the D input needed to be connected only to the /Q output, pin 6, so the flip-flop

would act as a toggle flip-flop. Second, the 74HC390 counter was missing a signal to the second stage clear bit, pin 14. This pin was left floating and I determined it was originally connected to the microcontroller for an enable/disable control of the sonar transmitter. I instead tied the signal to ground, thus always enabling the transmitter. Third, the 2N222 transistor was wired backwards. I simply rewired the transistor as seen in the Appendix A2.

The receiver circuitry also had two errors. First, V^- of the MAX266 programmable active filter was left floating and GND was wired to both the system ground and biased at 🗁V^+. Instead, I connected V^- to system ground and I biased GND at 🗁V^+ only. Second, the ground input to the 15K potentiometer was left floating. I wired it to ground.

BEHAVIORS

Odin has four behaviors: Avoid Obstacle, Object Detect, Find Dock, and Follow Dock. The code for these behaviors is shown in Appendix B1. Since Odin is a rectangular platform, it is necessary to reverse the motors to escape obstacles. Therefore, it is important Odin reacts to obstacles it finds while in reverse. I therefore implemented three motor states, “DRIVE”, “PARK”, and “REVERSE”, which the behaviors must use to send the correct reaction to the motors.

Avoid Obstacle uses the IR receivers to determine where obstacles are around Odin. It also uses the current state of the motors to determine the next state of the motors. When an obstacle is detected it enables the “ir_detect” bit that the arbitrator uses to enable the motor controls set by the process. Those motor controls are “ir_gear”, the next state of the motors, and “new_left_motor” and “new_right_motor”, the turning control of the motors.

Object Detect uses the bump sensors to determine if Odin has run into an object. If so, the process sets the next state of the motors with “new_gear” based on the current state of the motors. Then the process sets the “bump” bit, which tells the arbitrator that a bump was detected.

Find Dock is a simple behavior that puts Odin in a spin with the IR emitters turned off, looking for the IR emitters on WobbleHead. It continues the spin until the IR detection is directly behind Odin, where the charge jack is located. This behavior needs to be revised so it first moves Odin away from any obstacles it may currently be near.

Follow Dock uses the IR detection found by Find Dock to drive towards WobbleHead in reverse and keeps the IR detection of WobbleHead in the center of its rear receivers. It set the “ir_detect” bit and uses “new_left_motor” and “new_right_motor” to control the motors via the arbitrator. This behavior only works if there are no obstacles between Odin and WobbleHead.

WobbleHead has two behaviors: Follow Sonar and Follow IR. The code for these behaviors is shown in Appendix B2. WobbleHead sole task is to find Odin and point his IR emitters toward Odin, so when Odin’s charge gets low and Odin looks for WobbleHead, Odin will be able to detect WobbleHead’s IR emitters. Follow Sonar and Follow IR are not explicitly defined in the WobbleHead code.

Follow Sonar polls the sonar detectors for a detection. When a detection is found, the process turns the motors such that the front of WobbleHead faces the origin of the detection, unless the

sonar originated from the front of WobbleHead already. Then, Follow Sonar allows Follow IR to control the motors.

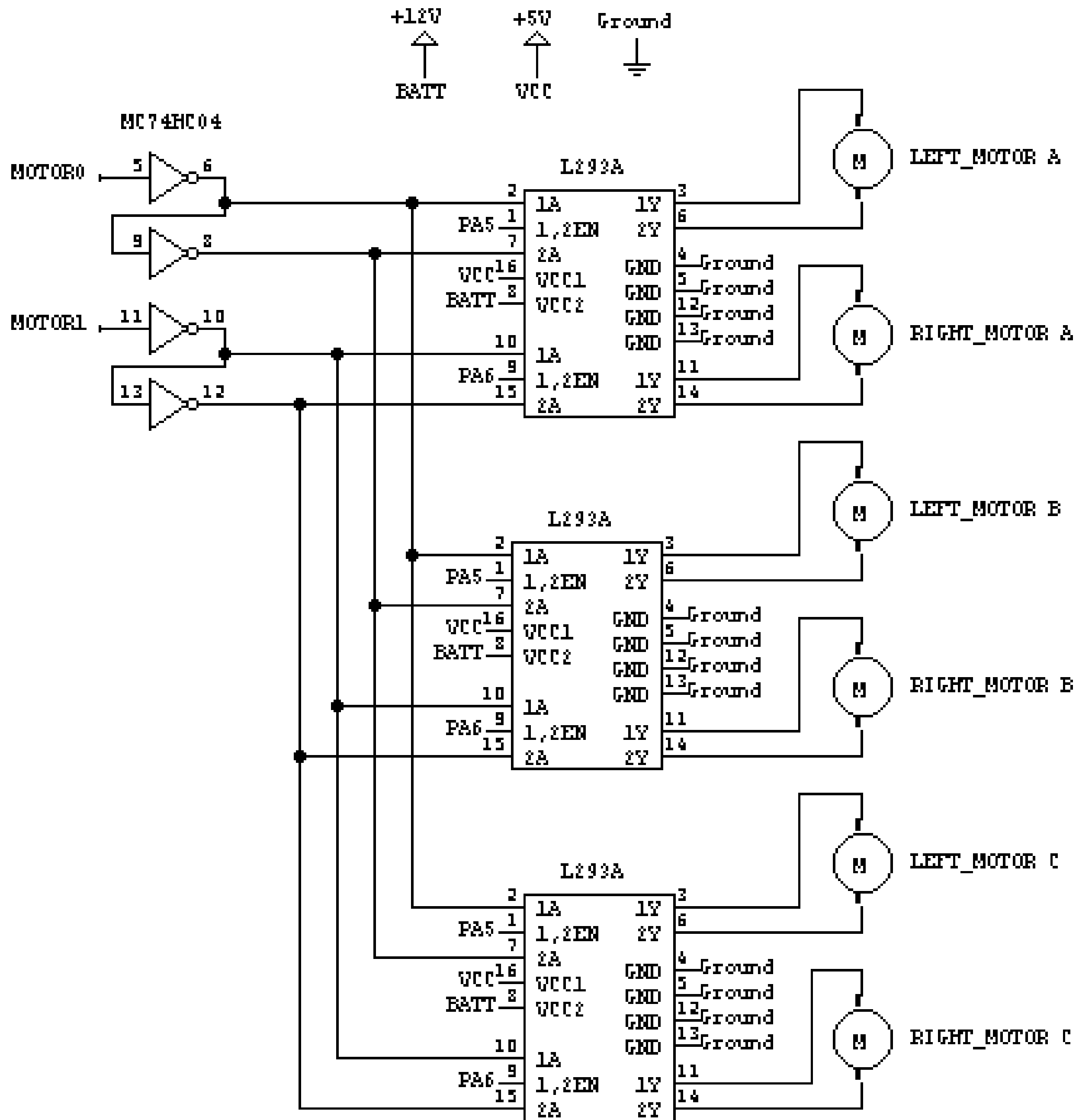
Follow IR uses the IR receivers to align the charge jack on WobbleHead with the charge jack on Odin. This behavior does not work correctly because the IR receivers are too close together to detect a difference in intensity of the IR signal. Ideally, two more IR receivers should be added to WobbleHead, one on each side of the present two, with a greater length between them. This behavior would then function properly.

CONCLUSION

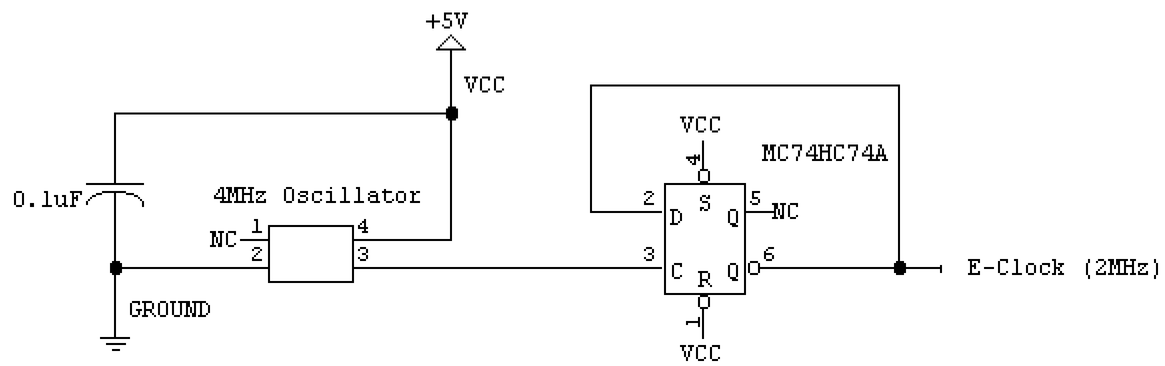
Odin was successful in avoiding obstacles while in all states of the motors. Odin was successful in finding the IR emitters of WobbleHead and driving toward them. WobbleHead was successful in finding the location of Odin with sonar. Odin and WobbleHead did not successfully dock. WobbleHead was unable to actuate the motors accurately or use the IR receivers to align with the charge jack of Odin. Odin was not able to make the small adjustments necessary for a successful dock. Ideally, Odin should have the ability to determine the distance to the docking station. This would enable Odin to know to make small adjustments when near WobbleHead. WobbleHead should have more IR receivers to accurately determine actuation at close range. However, the project successfully demonstrated the potential of using a charging station that could dynamically change its orientation to the charging robot.

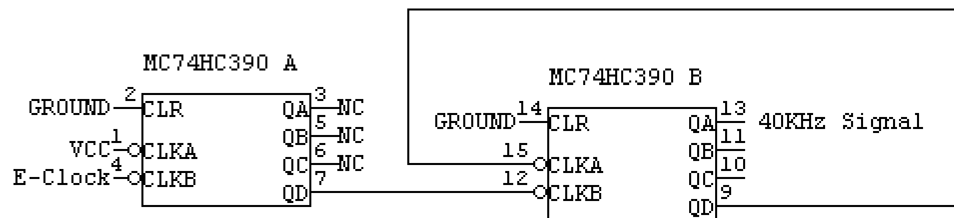
APPENDIX A: CIRCUITRY

A1: Motor Driver Circuitry



A2: E-Clock and 40KHz signal Generation Circuitry





APPENDIX B: SOFTWARE

B1: Odin Software

```
/*    Title: Global.c
      Programmer: Michael Apodaca
      Date: April 23, 1998
      Version: 1.5
      Description:
            This file defines all global contants, global
            variables, and function prototypes for the program
            Odin.c
*/
/* CONSTANTS */
#define SENSORS *(unsigned char *) 0xFFB8 /* address of latch     which controls
analog MUX */
#define IR *(unsigned char *) 0xFFB9            /* address of IR emitters */
#define ON 0xFF
#define OFF 0x00
#define TRUE 1
#define FALSE 0
#define IR_THRESHOLD 106
#define BUMP_THRESHOLD 10
#define PARK 0
#define REVERSE -70
#define DRIVE 70
#define FAST 20
#define SLOW 30
#define LOW_BATT 95
#define CHARGING 100

/* GLOBAL VARIABLES */
char start[]={0x1b,'[','3',';','1','H'};

/* Read_Sensor */
int front_left_wheel;
int front_left_body;
int front_right_body;
int front_right_wheel;
int rear_left_wheel;
int rear_left_body;
int rear_right_body;
int rear_right_wheel;
int side_right;
int side_left;
int battery = 0;
int rear_bump = 0;
int front_bump = 0;
int charge = 0;
int sensor_mirror;            /* mirrors SENSORS ($ffb8) since it's a read only
register */

/* Analog to Digital */
int detect_front_left_wheel = FALSE;
int detect_front_left_body = FALSE;
int detect_front_right_body = FALSE;
int detect_front_right_wheel = FALSE;
int detect_rear_left_wheel = FALSE;
int detect_rear_left_body = FALSE;
int detect_rear_right_body = FALSE;
```

```
int detect_rear_right_wheel = FALSE;
int detect_side_right = FALSE;
int detect_side_left = FALSE;
int battery_low = FALSE;
int charging_batt = FASLE;

/* Object Detect */
int bump = FALSE;
int new_gear = PARK;

/* Avoid_Obstacle */
int ir_detect = FALSE;
int ir_gear = PARK;
int new_left_motor = PARK;
int new_right_motor = PARK;

/* Arbitrator */
int gear = PARK;
int left_motor = PARK;
int right_motor = PARK;

/* PROTOTYPES */
void Initialize();
void Read_Sensors(char);
void Display_Sensors();
void Analog_to_Digital();
void Avoid_Obstacle();
void Object_Detect();
void Arbitrator();
void Return_Home();
void Find_Dock();
void Follow_Dock();
void Arbitrate();

/*    Title: Odin.c
      Programmer: Michael Apodaca
      Date: April 23, 1998
      Version: 2.1
      Description:
            This program implements Obstacle Avoidance and
            Find Charging Station algorithms for ODIN.  It
            also outputs all sensor values to Kermit/Terminal.
*/
#include <motor2l.h>
#include <serial.c>
#include <analog.h>
#include <vectors.h>
#include "Global.c"

/*----------------------------------------------------------------------*/
void main(){
      Initialize();

      while(1){
            Read_Sensors(ON);
            Analog_to_Digital();
            Object_Detect();
            Avoid_Obstacle();
            Display_Sensors();
        Arbitrator();
      }
```

```
}

/*----------------------------------------------------------------------*/

void Initialize(){
  init_motors();
  init_analog();
  init_serial();
}

/*----------------------------------------------------------------------*/
/* Read Sensors reads all the sensor values used throughout the program */
void Read_Sensors(char ir_value){
/* Set IR emitters on or off*/
  IR = ir_value;

/* read Battery and Bumpers while IR charges */
  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x10);
  SENSORS = sensor_mirror;

  charge = analog(0);
  rear_bump = analog(1);

  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x11);
  SENSORS = sensor_mirror;

  battery = analog(0);
  front_bump = analog(1);

/*  read IR values  */
  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x0f);
  SENSORS = sensor_mirror;

  rear_right_wheel = analog(2);                 /* IRDT1 */
  front_right_wheel = analog(3);          /* IRDT2 */
  rear_right_body = analog(4);            /* IRDT3 */
  front_left_wheel = analog(5);                 /* IRDT4 */
  rear_left_wheel = analog(6);                  /* IRDT5 */
  rear_left_body = analog(7);             /* IRDT6 */

  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x13);
  SENSORS = sensor_mirror;

  side_right = analog(0);                       /* IRDT8 */

  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x14);
  SENSORS = sensor_mirror;

  front_right_body = analog(0);                 /* IRDT9 */

  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x15);
  SENSORS = sensor_mirror;

  front_left_body = analog(0);                  /* IRDT10 */
```

```
  CLEAR_BIT(sensor_mirror, 0x1f);
  SET_BIT(sensor_mirror, 0x16);
  SENSORS = sensor_mirror;

  side_left = analog(0);                        /* IRDT11 */

}

/*----------------------------------------------------------------------*/
/* Display Sensors outputs the value of the sensors to a standard VT100
   Terminal program */
void Display_Sensors(){
      write(start);     /* Start at top of screen */

      write("Battery = ");
      put_int(battery);
      put_char(11);
      put_char(13);
      write("Charge = ");
      put_int(charge);
      put_char(11);
      put_char(13);
      write("Front Bumper = ");
      put_int(front_bump);
      put_char(11);
      put_char(13);
      write("Rear Bumper = ");
      put_int(rear_bump);
      put_char(11);
      put_char(13);
      write("Front Left Wheel = ");
      put_int(detect_front_left_wheel);
      put_char(11);
      put_char(13);
      write("Front Left Body = ");
      put_int(detect_front_left_body);
      put_char(11);
      put_char(13);
      write("Front Right Body = ");
      put_int(detect_front_right_body);
      put_char(11);
      put_char(13);
      write("Front Right Wheel = ");
      put_int(detect_front_right_wheel);
      put_char(11);
      put_char(13);
      write("Rear Left Wheel = ");
      put_int(detect_rear_left_wheel);
      put_char(11);
      put_char(13);
      write("Rear Left Body = ");
      put_int(detect_rear_left_body);
      put_char(11);
      put_char(13);
      write("Rear Right Body = ");
      put_int(detect_rear_right_body);
      put_char(11);
      put_char(13);
      write("Rear Right Wheel = ");
      put_int(detect_rear_right_wheel);
      put_char(11);
```

```
      put_char(13);
      write("Side Left = ");
      put_int(detect_side_left);
      put_char(11);
      put_char(13);
      write("Side Right = ");
      put_int(detect_side_right);
      put_char(11);
      put_char(13);

      write("Bump Detect = ");
      put_int(bump);
      put_char(11);
      put_char(13);
      write("Bump Gear = ");
      put_int(new_gear);
      put_char(11);
      put_char(13);
      write("IR Detect = ");
      put_int(ir_detect);
      put_char(11);
      put_char(13);
      write("IR Gear = ");
      put_int(ir_gear);
      put_char(11);
      put_char(13);
      write("IR Left Gear = ");
      put_int(left_motor);
      put_char(11);
      put_char(13);
      write("IR Right Gear = ");
      put_int(right_motor);
      put_char(11);
      put_char(13);
      write("Current Gear = ");
      put_int(gear);
      put_char(11);
      put_char(13);
}

/*---------------------------------------------------------------------------*/
/* Analog to Digital converts all analog input values (0 to 255) to a
   Digital value (True or False).  This is used so future modifications
   of sensor logic can be easily implemented.  For example, this function
   only uses the current sensor reading, however, in the future current
   values may be combined with previous values.
*/
void Analog_to_Digital(){

  if (rear_right_wheel > IR_THRESHOLD)
        detect_rear_right_wheel = TRUE;
  else
        detect_rear_right_wheel = FALSE;

  if (front_right_wheel > IR_THRESHOLD)
        detect_front_right_wheel = TRUE;
  else
        detect_front_right_wheel = FALSE;

  if (rear_right_body > IR_THRESHOLD)
        detect_rear_right_body = TRUE;
```

```
  else
        detect_rear_right_body = FALSE;

  if (front_left_wheel > IR_THRESHOLD)
        detect_front_left_wheel = TRUE;
  else
        detect_front_left_wheel = FALSE;

  if (rear_left_wheel > IR_THRESHOLD)
        detect_rear_left_wheel = TRUE;
  else
        detect_rear_left_wheel = FALSE;

  if (rear_left_body > IR_THRESHOLD)
        detect_rear_left_body = TRUE;
  else
        detect_rear_left_body = FALSE;

  if (side_right > IR_THRESHOLD)
        detect_side_right = TRUE;
  else
        detect_side_right = FALSE;

  if (front_right_body > IR_THRESHOLD)
        detect_front_right_body = TRUE;
  else
        detect_front_right_body = FALSE;

  if (front_left_body > IR_THRESHOLD)
        detect_front_left_body = TRUE;
  else
        detect_front_left_body = FALSE;

  if (side_left > IR_THRESHOLD)
        detect_side_left = TRUE;
  else
        detect_side_left = FALSE;

  if (battery < LOW_BATT)
        battery_low = TRUE;
  else
        battery_low = FALSE;

  if (charge > CHARGING)
        charging_batt = TRUE;
  else
        charging_batt = FLASE
}

/*--------------------------------------------------------------------------*/
/* Object Detect determines if a bumper was activated and then
   sends the appropriate motor command to the arbitrator based
   on the motors' current state.
*/
void Object_Detect(){
      bump = FALSE;
      if ((gear == DRIVE) && (front_bump > BUMP_THRESHOLD)){
            bump = TRUE;
            /* Shift Gears */
            new_gear = REVERSE;
      }
```

```
	else if ((gear == REVERSE) && (rear_bump > BUMP_THRESHOLD)){
		bump = TRUE;
		/* Shift Gears */
		new_gear = DRIVE;
	}
	else if ((gear == PARK) && (rear_bump > BUMP_THRESHOLD)){
		bump = TRUE;
		new_gear = DRIVE;
	}
}

/*-------------------------------------------------------------------------*/
/* Avoid Obstacle determines the appropriate motor reaction to the combination
   of the current state of the motors and the IR detector values
*/
void Avoid_Obstacle(){
	ir_detect= FALSE;
	if (gear == DRIVE){
		if (detect_front_left_body || detect_front_right_body){
			if (detect_front_left_wheel && detect_front_right_wheel){
				ir_detect= TRUE;
				/* Reverse direction and turn right */
				ir_gear = REVERSE;
				new_left_motor = DRIVE - SLOW;
				new_right_motor = REVERSE;
			}
			else if (detect_front_left_wheel){
				ir_detect= TRUE;
				/* Slow left and reverse right motors */
				ir_gear = DRIVE;
				new_left_motor = DRIVE - SLOW;
				new_right_motor = REVERSE;
			}
			else if (detect_front_right_wheel){
				ir_detect= TRUE;
				/* Slow right and reverse left motors */
				ir_gear = DRIVE;
				new_left_motor = REVERSE;
				new_right_motor = DRIVE - SLOW;
			}
			else if (detect_front_left_body && detect_front_right_body){
				if (detect_side_right){
					ir_detect= TRUE;
					/* Reverse direction and turn left */
					ir_gear = REVERSE;
					new_left_motor = REVERSE;
					new_right_motor = DRIVE - SLOW;
				}
				else if (detect_side_left){
					ir_detect= TRUE;
					/* Reverse direction and turn right */
					ir_gear = REVERSE;
					new_left_motor = DRIVE - SLOW;
					new_right_motor = REVERSE;
				}
				else {
					ir_detect= TRUE;
					/* Spin around */
					ir_gear = DRIVE;
					new_left_motor = REVERSE;
					new_right_motor = DRIVE;
```

```
                }
            }
        }
        else if (detect_front_left_wheel){
            ir_detect= TRUE;
            /* Slow left and reverse right motors */
            ir_gear = DRIVE;
            new_left_motor = DRIVE - SLOW;
            new_right_motor = REVERSE;
        }
        else if (detect_front_right_wheel){
            ir_detect= TRUE;
            /* Slow right and reverse left motors */
            ir_gear = DRIVE;
            new_left_motor = REVERSE;
            new_right_motor = DRIVE - SLOW;
        }
        else if (detect_side_right){
            ir_detect = TRUE;
            /* Increase right wheel to avoid wall */
            ir_gear = DRIVE;
            new_left_motor = DRIVE;
            new_right_motor = DRIVE + FAST;
        }
        else if (detect_side_left){
            ir_detect = TRUE;
            /* Increase left wheel to avoid wall */
            ir_gear = DRIVE;
            new_left_motor = DRIVE + FAST;
            new_right_motor = DRIVE;
        }
        else if (detect_rear_left_body && detect_rear_right_body){
            ir_detect= TRUE;
            /* Increase speed */
            ir_gear = DRIVE;
            new_left_motor = DRIVE + FAST;
            new_right_motor = DRIVE + FAST;
        }
        else if (detect_rear_left_wheel){
            ir_detect= TRUE;
            /* Increse left wheels */
            ir_gear = DRIVE ;
            new_left_motor = DRIVE + FAST;
            new_right_motor = DRIVE;
        }
        else if (detect_rear_right_wheel){
            ir_detect= TRUE;
            /* Increse right wheels */
            ir_gear = DRIVE;
            new_left_motor = DRIVE;
            new_right_motor = DRIVE + FAST;
        }
    }

    else if (gear == REVERSE){
        if (detect_rear_left_body || detect_rear_right_body){
            if (detect_rear_left_wheel && detect_rear_right_wheel){
                ir_detect= TRUE;
                /* Reverse direction and turn left */
                ir_gear = DRIVE;
                new_left_motor = REVERSE + SLOW;
```

```
                new_right_motor = DRIVE;
        }
        else if (detect_rear_left_wheel){
                ir_detect= TRUE;
                /* Slow left and reverse right motors */
                ir_gear = REVERSE;
                new_left_motor = REVERSE + SLOW;
                new_right_motor = DRIVE;
        }
        else if (detect_rear_right_wheel){
                ir_detect= TRUE;
                /* Slow right and reverse left motors */
                ir_gear = REVERSE;
                new_left_motor = DRIVE;
                new_right_motor = REVERSE + SLOW;
        }
        else if (detect_rear_left_body && detect_rear_right_body){
                if (detect_side_right){
                        ir_detect= TRUE;
                        /* Reverse direction and turn right */
                        ir_gear = DRIVE;
                        new_left_motor = DRIVE;
                        new_right_motor = REVERSE + SLOW;
                }
                else if (detect_side_left){
                        ir_detect= TRUE;
                        /* Reverse direction and turn left */
                        ir_gear = DRIVE;
                        new_left_motor = REVERSE + SLOW;
                        new_right_motor = DRIVE;
                }
                else {
                        ir_detect= TRUE;
                        /* Spin around */
                        ir_gear = REVERSE;
                        new_left_motor = REVERSE;
                        new_right_motor = DRIVE;
                }
        }
}
else if (detect_rear_left_wheel){
        ir_detect= TRUE;
        /* Slow left and reverse right motors */
        ir_gear = REVERSE;
        new_left_motor = REVERSE + SLOW;
        new_right_motor = DRIVE;
}
else if (detect_rear_right_wheel){
        ir_detect= TRUE;
        /* Slow right and reverse left motors */
        ir_gear = REVERSE;
        new_left_motor = DRIVE;
        new_right_motor = REVERSE + SLOW;
}
else if (detect_side_right){
        ir_detect = TRUE;
        /* Increase right wheel to avoid wall */
        ir_gear = REVERSE;
        new_left_motor = REVERSE;
        new_right_motor = REVERSE - FAST;
}
```

```
            else if (detect_side_left){
                  ir_detect = TRUE;
                  /* Increase left wheel to avoid wall */
                  ir_gear = REVERSE;
                  new_left_motor = REVERSE - FAST;
                  new_right_motor = REVERSE;
            }
            else if (detect_front_left_body && detect_front_right_body){
                  ir_detect= TRUE;
                  /* Increase speed */
                  ir_gear = REVERSE;
                  new_left_motor = REVERSE - FAST;
                  new_right_motor = REVERSE - FAST;
            }
            else if (detect_front_left_wheel){
                  ir_detect= TRUE;
                  /* Increase left wheels */
                  ir_gear = REVERSE;
                  new_left_motor = REVERSE - FAST;
                  new_right_motor = REVERSE;
            }
            else if (detect_front_right_wheel){
                  ir_detect= TRUE;
                  /* Increse right wheels */
                  ir_gear = REVERSE;
                  new_left_motor = REVERSE;
                  new_right_motor = REVERSE - FAST;
            }
      }
}

/*-------------------------------------------------------------------------*/
/* Arbitatrator arbitrates the motor commands from the
   highest priority task
*/
void Arbitrator(){
      int i, count;
      /* Highest priority is Object_Detect */
      for (i = 0; i <= 10; i++){
            for (count = 10; count >= 0; count--){
                  gear = (new_gear + (count*gear))/(count+1);
                  motor(0,gear);
                  motor(1,gear);
            }
      }
      gear = new_gear;

      /* if battery is low then dock */
      if (battery_low){
            Return_Home();
      }

      /* else Avoid Obstacles */
      else if (ir_detect && !bump){
            for (i = 0; i <= 10; i++){
                  for (count = 10; count >= 0; count--){
                        left_motor = (new_left_motor +
(count*left_motor))/(count+1);
                        right_motor = (new_right_motor +
(count*right_motor))/(count+1);
                        motor(0, left_motor);
```

```
                        motor(1, right_motor);
                  }
            }
            gear = ir_gear;
            new_gear = ir_gear;
      }
}

/*--------------------------------------------------------------------------*/

void Return_Home(){
/* Face Rear Towards IR */
      Find_Dock();
/* Follow IR until reach Charger */
      while (! charging_batt){
/* Read Sensors */
            Read_Sensors(ON);
/* Digital Conversion */
            Analog_to_Digital();
/* Follow IR */
            Follow_Dock();
/* Bump Detect */
            Object_Detect();
/* Output Sensors to SCI */
            Display_Sensors();
/* Arbitrate */
            Arbitrate();
      }
/* Wait for Batteries to Charge */
      Charge_Batteries();
}

/*--------------------------------------------------------------------------*/
/* Find Dock reads the IR sensors and spins Odin until the charge
   jack in the rear faces toward the docking station */
void Find_Dock(){
/* Turn off IR emmiters and Read Sensors */
      Read_Sensors(OFF);
      Analog_to_Digital();
/* Spin around until charge jack in rear faces charging station */
      while (! (detect_rear_left_body && detect_rear_right_body)){
            motor(0, REVERSE);
            motor(1, DRIVE);
            Read_Sensors(OFF);
            Analog_to_Digital();
      }
}

/*--------------------------------------------------------------------------*/
/* Follow Dock uses the IR values to steer towards the charging station. */
void Follow_Dock(){
      ir_detect = FALSE;
      if (detect_rear_left_body || detect_rear_right_body){
            if (detect_rear_left_wheel && detect_rear_right_wheel){
                  ir_detect= TRUE;
                  /* Back straight towards */
                  ir_gear = REVERSE;
                  new_left_motor = REVERSE;
                  new_right_motor = REVERSE;
            }
            else if (detect_rear_left_wheel){
```

```
                ir_detect= TRUE;
                /* Increase left wheels */
                ir_gear = REVERSE;
                new_left_motor = REVERSE + FAST;
                new_right_motor = REVERSE;
            }
            else if (detect_rear_right_wheel){
                ir_detect= TRUE;
                /* Increase right wheels */
                ir_gear = REVERSE;
                new_left_motor = REVERSE;
                new_right_motor = REVERSE + FAST;
            }
            else if (detect_rear_left_body && detect_rear_right_body){
                ir_detect= TRUE;
                /* Back straight towards */
                ir_gear = REVERSE;
                new_left_motor = REVERSE;
                new_right_motor = REVERSE;
            }
        }
        else if (detect_rear_left_wheel){
            ir_detect= TRUE;
            /* Turn towards right */
            ir_gear = REVERSE;
            new_left_motor = REVERSE - FAST;
            new_right_motor = REVERSE + SLOW;
        }
        else if (detect_rear_right_wheel){
            ir_detect= TRUE;
            /* Turn towards left */
            ir_gear = REVERSE;
            new_left_motor = REVERSE + SLOW;
            new_right_motor = REVERSE - FAST;
        }
        else {
            /* Charging station not found */
            lost_station = TRUE;
        }
}

/*-------------------------------------------------------------------------*/
/* Arbitrate arbitrates the motor controls from the highest
   priority task.
*/
void Arbitrate(){
        int i, count;
        /* Highest priority is Object_Detect */
        for (i = 0; i <= 10; i++){
            for (count = 10; count >= 0; count--){
                gear = (new_gear + (count*gear))/(count+1);
                motor(0,gear);
                motor(1,gear);
            }
        }
        gear = new_gear;

        /* Avoid Obstacles */
        if (ir_detect && !bump){
            for (i = 0; i <= 10; i++){
                for (count = 10; count >= 0; count--){
```

```
                        left_motor = (new_left_motor +
(count*left_motor))/(count+1);
                        right_motor = (new_right_motor +
(count*right_motor))/(count+1);
                        motor(0, left_motor);
                        motor(1, right_motor);
                  }
            }
            gear = ir_gear;
            new_gear = ir_gear;
      }

      /* Find lost charging station */
      else if (lost_station){
            Find_Dock();
      }
}

/*--------------------------------------------------------------------------*/
/* Charge Batteries turns off the motors and waits in a loop.
   There is no sensor for determining when the battereis are
   charged, so this procedure is inefficient.
*/
void Charge_Batteries(){
      motor(0, PARK);
      motor(1, PARK);

      int i,j;
      for (i = 0; i < 1000; i++)
            for (j=0; j < 1000; j++);
}

/*--------------------------------------------------------------------------*/
```

B2: WobbleHead Software

```
/*    Title: WobbleHead.c
      Programmer: Michael Apodaca
      Date: April 23, 1998
      Version: 1.1
      Description:
            This program implements the alignment algorithms for the
            charging station WobbleHead using sonar and IR sensors .
*/
#include <hc11.h>
#include <mil.h>
#include <vectors.h>
#include <analog.h>
#include <serial.c>

/*------------------------------------------------------------------------*/

/* IR Controls */
#define LEFT_IRDT 6            /* PortE Bit6 */
#define RIGHT_IRDT 7     /* PortE Bit7 */
#define IR_THRESHOLD 100
#define VARIANCE 5

/* IR Emitter Control */
#define IR PORTB         /* Bits 3,4,5 */

/* Sonar Sensors */
#define SONAR PORTA
#define SONAR_FRONT 1    /* PortA Bit1 */
#define SONAR_BACK 2     /* PortA Bit2 */
#define SONAR_LEFT 3     /* PortA Bit3 */
#define BITF 0x02
#define BITB 0x04
#define BITL 0x08

/* Motor Controls */
#define MOTOR PORTB
#define DIRECTION_BIT 0  /* PortB Bit0 */
#define CONTROL_BIT 1    /* PortB Bit1 */

/*------------------------------------------------------------------------*/

/* Global Values */
int left_ir, right_ir;
char front_sonar, back_sonar, left_sonar;
char port_a;

/*------------------------------------------------------------------------*/

void Init_Motor(){
      CLEAR_BIT(MOTOR, CONTROL_BIT);
      CLEAR_BIT(MOTOR, DIRECTION_BIT);
}

/*------------------------------------------------------------------------*/
/* Run motor toggles the control bit at 50% PWM.
*/
void Run_Motor(){
      int i, j;
```

```
	for (i = 0; i < 5000; i++){
		for (j = 0; j < 10; j++)
			SET_BIT(MOTOR, CONTROL_BIT);

		for (j = 0; j < 10; j++)
			CLEAR_BIT(MOTOR, CONTROL_BIT);
	}
}
/*-----------------------------------------------------------------------*/

void Stop_Motor(){
	CLEAR_BIT(MOTOR,CONTROL_BIT);
}

/*-----------------------------------------------------------------------*/

void Motor_Left(){
	CLEAR_BIT(MOTOR,DIRECTION_BIT);				/* LEFT */
	SET_BIT(MOTOR,CONTROL_BIT);				/* ON */
	Run_Motor();
}

/*-----------------------------------------------------------------------*/

void Motor_Right(){
	SET_BIT(MOTOR,DIRECTION_BIT);			/* RIGHT */
	SET_BIT(MOTOR,CONTROL_BIT);				/* ON */
	Run_Motor();
}

/*-----------------------------------------------------------------------*/

void Read_IR(){
	/* modulates PORTB, bit 3,4,5 at 40kHz */
	asm("ldaa    4100\n"
		"ldy    #255\n"		/* 2*(# of IR pulses) */
		"eora #56\n"
		"staa    4100\n");	/* 4099 = Port C */
	asm("loop1 : ldaa 4100\n"	/* 4 cycles - necessary */
		"eora #56\n"		/* 2 cycles - necessary */
		"staa 4100\n"		/* 4 cycles - necessary */
		"nop\n"
		"nop\n"
		"nop\n"				/* 8 cycles */
		"next1 : nop");
	asm("dey\n"			/* 4 cycles - necessary */
		"bne loop1");		/* 3 cycles - necessary */
						/* total = 25 cycles = 40 kHz*/
	left_ir = analog(LEFT_IRDT);

	/* modulates PORTB, bit 3,4,5 at 40kHz */
	asm("ldaa    4100\n"
		"ldy    #255\n"		/* 2*(# of IR pulses) */
		"eora #56\n"
		"staa    4100\n");	/* 4099 = Port C */
	asm("loop2 : ldaa 4100\n"	/* 4 cycles - necessary */
		"eora #56\n"		/* 2 cycles - necessary */
		"staa 4100\n"		/* 4 cycles - necessary */
		"nop\n"
		"nop\n"
		"nop\n"				/* 8 cycles */
```

```
            "next2 : nop");
      asm("dey\n"                 /* 4 cycles - necessary */
            "bne loop2");           /* 3 cycles - necessary */
                                                /* total = 25 cycles = 40 kHz*/
      right_ir = analog(RIGHT_IRDT);
}

/*------------------------------------------------------------------------*/

void Init_Sonar(){
      front_sonar = BITF;
      back_sonar = BITB;
      left_sonar = BITL;
}

/*------------------------------------------------------------------------*/
/* Read Sonar polls the sonar bits until a detection is found */
void Read_Sonar(){
      int i;
      while ((front_sonar == BITF) || (back_sonar == BITB) || (left_sonar ==
BITL)){
            front_sonar &= SONAR & BITF;
            back_sonar &= SONAR & BITB;
            left_sonar &= SONAR & BITL;
      }
}

/*------------------------------------------------------------------------*/
/* Arbitrator controls the motor with sonar unless the sonar reading
   originates from the front where it uses IR to control the motor
*/
void Arbitrator(){
      if (front_sonar != BITF){
            if ((left_ir > IR_THRESHOLD)&&(right_ir > IR_THRESHOLD)){
                  if (left_ir > (right_ir + VARIANCE)){
                        Motor_Left();
                  }
                  else if (right_ir > (left_ir + VARIANCE)){
                        Motor_Right();
                  }
                  else{
                        Stop_Motor();
                  }
            }
            else if (left_ir > IR_THRESHOLD){
                  Motor_Left();
            }
            else if (right_ir > IR_THRESHOLD){
                  Motor_Right();
            }
            else{
                  Motor_Left();
            }
      }
      else if (back_sonar != BITB){
            Motor_Left();
      }
      else if (left_sonar != BITL){
            Motor_Left();
      }
      else{
```

```
            Motor_Right();
      }
}

/*---------------------------------------------------------------------*/

void Show_Values(){
      char start[] = {0x1b,'[',';','1','H'};
      write(start);

      write("Left IR = ");
      put_int(left_ir);
      put_char(11);
      put_char(13);
      write("Right IR = ");
      put_int(right_ir);
      put_char(11);
      put_char(13);
      write("Front Sonar = ");
      put_int((int)front_sonar);
      put_char(11);
      put_char(13);
      write("Rear Sonar = ");
      put_int((int)back_sonar);
      put_char(11);
      put_char(13);
      write("Left Sonar = ");
      put_int((int)left_sonar);
      put_char(11);
      put_char(13);
      write("PORTA = ");
      put_int((int)port_a);
      put_char(11);
      put_char(13);
}

/*---------------------------------------------------------------------*/

void main(){
      Init_Motor();
      Init_Sonar();
      init_analog();
      init_serial();

      while(1){
            Read_Sonar();
            Read_IR();
            Arbitrator();
            Show_Values();
      }
}
/*---------------------------------------------------------------------*/
```

B3: Odin Motor Software Drivers

```
/* Title          motor.c
 * Programmer     Keith L. Doty. modified by Lee Rossey
 * Date           Oct 19, 1996
 * Version  1.1
 * Description
 *    This module includes motor initialization, motor speed control
 *    and two PWM interrupt drivers motor0 and motor1.
 *    motor0 uses OC2 and motor1 uses OC3.
 *    This module can be used with the TALRIK robot
 *
 */

/**************** Constants *************************************/
#include <motor2l.h>
#define PERIODM 65,500
#define PERIOD_1PC 655
#define SENSORS *(unsigned char *) 0xFFB8
/*  #define SENSORS *(unsigned char*) 0xffb8  */
#pragma interrupt_handler motor0 motor1

   void motor0();
   void motor1();
   void sensor_bit_clear();
   void sensor_bit_set();

/********************* Data *************************************/

   int duty_cycle[2];  /* Specifies the PWM duty cycle for two motors */
   extern int sensor_mirror;  /* mirrors SENSORS ($ffb8) */

/************** Functions ***************************************/
void init_motors(void)
/* Function: This routine initializes the motors
 * Inputs:   None
 * Outputs:  None
 * Notes:    This routine MUST be called to enable motor operation!
 */

{

  INTR_OFF();

/* Set OC2 and OC3 to output low */
  SET_BIT(TCTL1,0xA0);
  CLEAR_BIT(TCTL1,0x50);

/* Set PWM duty cycle to 0 first */
   duty_cycle[0] = duty_cycle[1] =0;

/* Associate interrupt vectors with motor routines */
  *(void(**)())0xFFE6 = motor0;
  *(void(**)())0xFFE4 = motor1;

/* Enable motor interrupts on OC2 and OC3 */
  SET_BIT(TMSK1,0x60);


/* Specify PD4 and PD5 as output pins.
```

```
 * PD4 controls direction of Motor 1 and PD5 the direction of Motor 0.
 */

/*  SET_BIT(DDRD,0x30); */
  INTR_ON();
}

void motor(int index, int per_cent_duty_cycle)
/* Function: Sets duty cycle and direction of motor specified by index
 * Inputs:   index in [0,1]
 *         -100% <= per_cent_duty_cycle <= 100%
 *         A negative % reverses the motor direction
 * Outputs:  duty_cycle[index]
 *           0 <= duty_cycle[index]<= PERIOD (Typically, PERIOD = 65,500)
 * Notes:    Checks for proper input bounds
 */
{
  if (per_cent_duty_cycle < 0)
     {
        per_cent_duty_cycle = -per_cent_duty_cycle; /* Make positive */
     /* Set negative direction of motors */
        if (index == 1) sensor_bit_clear(0x40);
        if (index == 0) sensor_bit_clear(0x80);
     }
  else
     {
     /* Set positive direction of motors */
        if (index == 1) sensor_bit_set(0x40);
        if (index == 0) sensor_bit_set(0x80);

     }

/* At this point per_cent_duty_cycle must be a positive number less
 * than 100. If not make it so.
 */
  if (per_cent_duty_cycle > 100) per_cent_duty_cycle = 100;
  duty_cycle[index] = per_cent_duty_cycle*PERIOD_1PC;

}

void motor0 ()
/* Function: This interrupt routine controls the PWM to motor0 using OC2
 * Inputs:   duty_cycle[0] (global)
 * Outputs:  Side effects on TCTL1, TOC2, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
/* Keep the motor off if no duty cycle specified.*/
  if(duty_cycle[0] == 0)
   {
    CLEAR_BIT(TCTL1, 0x40);
   }
  else
    if(TCTL1 & 0x40)
     {
      TOC2 += duty_cycle[0];                  /* Keep up for width */
      CLEAR_BIT(TCTL1,0x40);                  /*  Set to turn off */
     }
    else
     {
```

```
        TOC2 += (PERIODM - duty_cycle[0]);
        SET_BIT(TCTL1,0x40);                  /* Set to raise signal */
      }
  CLEAR_FLAG(TFLG1,0x40);                     /* Clear OC2F interrupt Flag */
}

void motor1()
/* Function: This interrupt routine controls the PWM to motor1 using OC3
 * Inputs:   duty_cycle[1] (global)
 * Outputs:  Side effects on TCTL1, TOC2, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
/* Keep the motor off if no duty cycle specified.*/
  if(duty_cycle[1] == 0)
   {
    CLEAR_BIT(TCTL1, 0x10);
   }
  else
    if(TCTL1 & 0x10)
      {
       TOC3 += duty_cycle[1];                 /* Keep up for width */
       CLEAR_BIT(TCTL1,0x10);                 /*  Set to turn off */
      }
    else
      {
       TOC3 += (PERIODM - duty_cycle[1]);
       SET_BIT(TCTL1,0x10);                   /* Set to raise signal */
      }
  CLEAR_FLAG(TFLG1,0x20);                   /* Clear OC3F interrupt Flag */
}


void sensor_bit_clear(int bit_clear)
{
     INTR_OFF();
     CLEAR_BIT(sensor_mirror, bit_clear);
     SENSORS = sensor_mirror;
     INTR_ON();
}

void sensor_bit_set(int bit_set)
{
     INTR_OFF();
     SET_BIT(sensor_mirror, bit_set);
     SENSORS = sensor_mirror;
     INTR_ON();
      }
```