Wiley Ammons
EEL5666
Final Written Report
December 4, 2001
Zircon: Detector of Studs

Table of Contents

Chapter page

ABSTRACT

I built a robot by the name of *Zircon*. The purpose of this machine is to locate studs within walls and mark them for the purposes of planning modifications to a pre-built room. A number of sensory systems were implemented to allow the robot to inspect its surroundings. These systems include IR, stud detection, and bump sensors. All of these sensors along with programs perform in tandem to make the robot operate and fulfill its intended function.

EXECUTIVE SUMMARY

I built this robot in order to fulfill the requirements of EEL5666 Intelligent Machine Design Lab (IMDL) and to satisfy a personal longing to get something concrete and practical out of my time here at college. The sensors included in the design were those suggested to me by the IMDL staff and faculty, and one of my own creation in interfacing the stud finder with my robot. I wrote the software for the robot in ICC11, which uses a low-feature version of C along with a compiler for the Motorola HC11. The HC11 powered the brains behind the robot and handled the inputs and outputs. The HC11 sat in a board called the MRC11 designed by Mekatronix (http://www.mekatronix.com). Expanding the input/output capabilities of the MRC11 was the MRSX01 another Mekatronix board available on their website. Mekatronix also has drivers written for servo, motor, and sensor control which can be easily obtained when you buy a robot kit from them. The behaviors programmed using this equipment consisted of wall following, obstacle avoidance, stud detection, floor marking, and calibration. The robot in its final form adheres to the objectives of the project as far as the limitations of the hardware allow. The servos have been hacked to provide speed-controlled DC motors, because of this, they are not evenly matched and do not keep the robot on a straight path when given the same pulse width duty cycle.

INTRODUCTION

After working a summer session with Miller Electric in Jacksonville, FL I gained a lot of experience in the electrical portion of the construction trade. When I took EEL 5666 as a course this fall, I remembered something that happened while we were working on a building in Vistakon that summer. We needed to add some light switches to a room, and I was to find the place to add them. I was then to cut a hole in the wall to make way for the switch and to place the box in the wall. When I found what I thought was a stud, I sawed a large hole in the wall. After sawing a gaping hole in the wall and making a plaster dust mess, I realized that what I thought was a stud, was simply a hard spot on the wall. I missed the stud by about 6 inches. The choice for my project seemed obvious.

I set out to build an autonomous agent that would ease the job of finding studs. The robot should also mark the floor beside each of the studs to leave a lasting indication and to make sure that there are no studs overlooked. Valuable information on the setup and operation of the microcontroller circuitry was provided by Mekatronix. Their manuals are well detailed and may be found on their website mentioned earlier in the Executive Summary.

CHAPTER 1

INTEGRATED SYSTEM

I built *Zircon* with an airplane-wood frame. The wood is light, strong, and readily available. The two-layer frame allows better support for the necessary circuitry. On the bottom layer of the robot, there are two servo motors, the wheels, the microcontroller boards, an IR sensor, a bump switch, and the batteries.

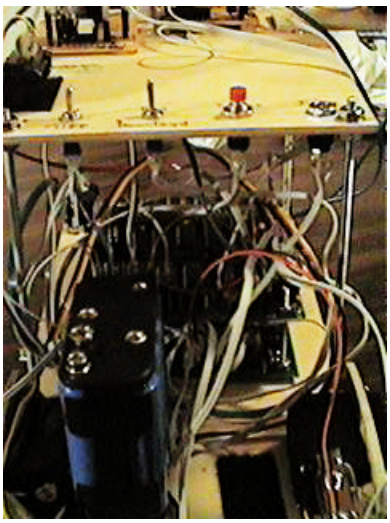


Figure 1: The bottom layer

On the top side of the upper layer a retail stud finder, an IR sensor, a charge jack, a solenoid, control circuitry for interfacing the sensors, and a system of switches for external control of the robot.

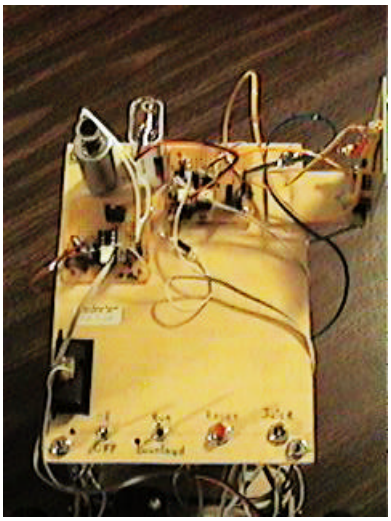


Figure 2: The upper layer

The robot has three modes: programming, basic obstacle avoidance, and stud detection/recognition.

Programming is the mode in which I load compiled code into robot RAM. Without this mode the robot will simply boot and start jerking around depending on the random data that appears on the ports when there is no code in the microcontroller.

Basic obstacle avoidance will be for demonstration purposes outside of the specified function of the robot. In this mode, the robot will move about a room avoiding obstructions, and reacting to bumps. I intended this mode to demonstrate the robot for an audience without very specific conditions of operation(walls). In order to operate in the obstacle avoidance mode the code programmed into the robot has to change, and the two IR sensors will need to be moved from their side-looking positions to forward-looking positions. For this reason, the IR sensors attach to the robot body with velcro. If necessary, they can be removed and relocated to the front.

The final mode, stud recognition, demonstrates the intended function of the robot.

This requires a set number of conditions to be met for the robot to perform effectively and as programmed. In this mode, the robots follows the wall it calibrates to and detects the studs behind the wall, marking the floor next to each stud. These conditions will not always be readily available, which lends to the creation of the obstacle avoidance mode.

CHAPTER 2

MOBILE PLATFORM

As I mentioned in the previous chapter, the robot has two levels, supported by four pieces of threaded rod, secured with nuts and washers. The threaded rods, nuts, and washers were purchased from Lowe's hardware. I used #6 rod because I had some extra nuts lying around from a previous project and saving money is a major consideration in IMDL. Threaded rod can be difficult to use at times. First, make sure that all of the holes drilled in the robot body are even and in correct places. Accurate measurement is of paramount importance when planning support structure. Another problem I happened upon is the fact that threaded rod left in a hardware too long may rust. Nuts are extremely difficult to screw onto a rusty piece of threaded rod. Sometimes a set of pliers cannot put the nut on without destroying the rod in the process. Lowe's stock rose during the time I worked on my project, and I can help but feel at least partially responsible.

I placed two wheels near the rear of the robot to drive and steer it. On the front of the robot, I placed a simple plastic stabilizer (stabilizer, used here, is a fancy phrase for a ping-pong ball with the top cut off). I glued the stabilizer to the bottom of the lower panel to hold the front up and keep the robot level while scanning the wall. I chose a stabilizer of this type because of the fact that a caster wheel would either require steering or depend on a free-spinning wheel. If a caster wheel ever turns sideways, the speed and force of the forward motion of the robot could break the wheel off. A broken caster would be a terrible problem to have on demo day, so I decided to avoid it.

I carefully considered the placing of the circuitry to keep the robot balanced. I tried to keep the weight on the back as much as possible to prevent use of the stabilizer up front as much as possible. There are things that can go wrong when the robot is resting all or in part on a roller that it can't control. Along with this, sand and dirt becomes a major consideration when it starts to wear away at a stabilizer that cannot be easily replaced. If taken to practical production the choice of material for the stabilizer would need to be changed. It lacks the durability for regular or heavy duty use.

At the last minute, the stud finder needed to be moved out onto a platform to make sure it stays close to the wall. In preliminary tests, the stud finder was able to detect the stud from a good distance. Once the robot ran along a wall with the stud finder mounted, it became clear that the distance from the wall was too great. I cut a small square of wood and secured it to the upper platform extending the stud finder more towards the wall (see figure 2, top right corner). More about the stud finder to follow in chapter 4.

CHAPTER 3

ACTUATION

The only actuation required for this robot design is a simple solenoid to control the marker mounted on the front of the robot and servos to move it around.

The solenoid is a 9 volt, 3 amp push-type tube solenoid from Magnetic Sensor Systems (http://www.solenoidcity.com). The solenoid itself costs $24.99 + shipping which ended up being $38.00 total. If you need the solenoid to return to an idle position make sure to spend *another* $1.00 to get the shaft return spring. The wiring on the solenoid cannot be done wrong, a convenient fact since the wires are not marked. **IMPORTANT NOTE**: This solenoid draws 3 full amps, a 9 volt battery will not run it. The power supplies in the IMDL lab will not run it. I spent several hours trying to figure out what was wrong with my solenoid until Scott took it to a power supply that could deliver that kind of current. This was a nice thing to find out, but as a safety measure, I had already spent another $38.00 to order a new one so as to have it in time for demo day. Expensive mistakes like this should be avoided as much as possible.

I glued a sharpee™ black permanent marker to the shaft of the solenoid to make the mark. The solenoid shaft and length of the marker was measured exactly so that when the solenoid plunges, the marker tips contacts the ground. When the solenoid releases, the return spring raises the pen to stop the marking process. I mounted the solenoid by measuring the size of the solenoid and drilling a hole in the upper platform that was just smaller. The business end of the solenoid is threaded, and I simply screwed the solenoid

into the wood through the hole I drilled.

I placed the servos at the rear of the robot mounted to the wood with the servo mounts designed by Mekatronix and taken from the TALRIK™ robot design. The servo mounts were secured to the body with Elmer's Extra Strength Wood Glue from Home Depot (http://www.homedepot.com).

The servos were purchased from Mekatronix and hacked down to gear head speed-controllable DC motors. This was done by routing out the half moon slot in the drive gear with a dremel tool. This prevents the gear from turning the potentiometer when the servo spins. If the potentiometer does not move, the servo will try to position itself forever, moving the robot. I also cut the stop off of the gear to prevent the servo from using a physical stop to keep it from moving. Make sure to super-glue the potentiometer in place, or the servo will find some way to adjust the potentiometer and ruin the calibration you worked so hard to obtain. The servos receive power from a secondary battery pack I use to power them and the solenoid. They plug into the sensor expansion board servo positions which connect the servo power connectors and ports from the microcontroller that generate the pulse width to the servo connector.

CHAPTER 4

SENSORS

The sensors I use on my robot are IR detectors, bump sensors, and a sensor mounted on the top of the robot to detect studs.

The IR sensors were purchased from Mekatronix and operate on a 40kHz signal. The signal is sent to the analog port where the robot uses the feedback from these sensors to maintain the adequate and necessary distance from the wall for stud detection. The IR sensors may be relocated to the front for the obstacle avoidance function as well. Otherwise the robot will simply have to bump into something to know that it is there.

The bump sensor was mounted on the rear, and lets the robot know it has aligned itself in a corner and is ready to perform its function.

The stud detector is a retail construction stud detector, hacked to provide a signal to the robot instead of to a human observer. I tapped the power and ground from the LED on the stud sensor that indicated a stud is found. I then ran both leads to the input side of an opto-isolator (given to me by Scott Nortman) and the output side of the opto-isolator connected a 5V power pin to one of the analog pins on the sensor expansion board. When the stud sensor detects a stud, 5V is sent to an analog pin, brining the value to 255. The robot then used this value to decide when to fire the solenoid.

Sensor Specification: IR

IR is a mode of sensor that has been used regularly by the IMDL for years. Because of this, no specification for IR is necessary.

Sensor Specification: Bump

Bump switches follow the same rule for specification as IR. They are so common that they need no special explanation. They are contact switches and voltage dividers.

Sensor Specification: Stud Detector

The stud detector is the primary sensor required for my robot to perform its function. My entire design is built around having a stud detector be carried along a wall and function normally.

In trying to discover the theory behind the stud detector, I have run into some problems. From what I can see, the stud detector uses three "panels" of conducting material on the bottom side of the circuit board. There is a signal sent to the large inside panel and according to what is in the wall in front of the panels a return signal comes to the two smaller side panels. You calibrate the sensor by putting it on blank wall and then moving it to discover a stud. There is an IC on the circuit board that I have not had the time to put a scope on to try and figure out what it does. Zircon, the company that makes it, holds a patent on the method of detection so it is probably a bit different than the other models out there. I think that the signal sent by the panel *may* be a magnetic field. In which case the IC controls the transmission and detection of the magnetic field. The large panel then would be a generator and the two side panels would detect the field strength to decide if there is something there. I have yet to test this with any sort of compass or other field detection device, I am speculating based on the data I have gotten from putting my voltage tester on leads on the circuit board.

The objective of integrating my stud detector into the robot is to be able to easily detect studs behind the wall. I thought about trying to design my own stud detector, and I am

quite happy now that I did not attempt it. The integration of the retail Zircon has been hard enough and I think it beats any of my designs in reliability.

The stud detector can be purchased from Home Depot, on Tower Rd. here in Gainesville, or from Zircon directly at www.zircon.com. It cost me $29.99 + tax at Home Depot, but the price on it has dropped since I bought it, which doesn't make me a happy person. As a side note, the brand name of the stud finder is where my robot got its name. Since the robot's mission in life is to carry around a Zircon stud finder, I named it that way as a thank you to Zircon for making my life easier in IMDL.

The full information on the stud detector is as follows:

Manufacturer: Zircon

Model: Studsensor Pro Spotlite

Model Number: 58148

Description:

- SpotLite Pointing System locates edges of studs with bright beam of light
- DeepScan mode for extra depth (1 1/2 inch) of drywall, plywood, paneling,
- For most bare wood flooring
- Patented Trucal instant calibration system to eliminate errors
- Audio and LED signal also indicates stud edge
- Modern contoured case design allows ease in use
- 9 volt battery provided
- Excellent for hanging pictures, mirrors, drapery rods, shelves
- Also for heavy objects which require fastening to a stud

Sell Pack Description: Each

Sell Pack Ship Weight:.25 oz.

Sell Pack Length: 4"

Sell Pack Width: 1.309999"

Sell Pack Height: 8.5"

Sell Pack Cubic Size: 44.54 in.^{3}

Sensor Effective On: Sheet rock,drywall,plywood, paneling, wood floor

Sound Indicator: Yes

Led Display: Yes

Automatic Operation: Yes

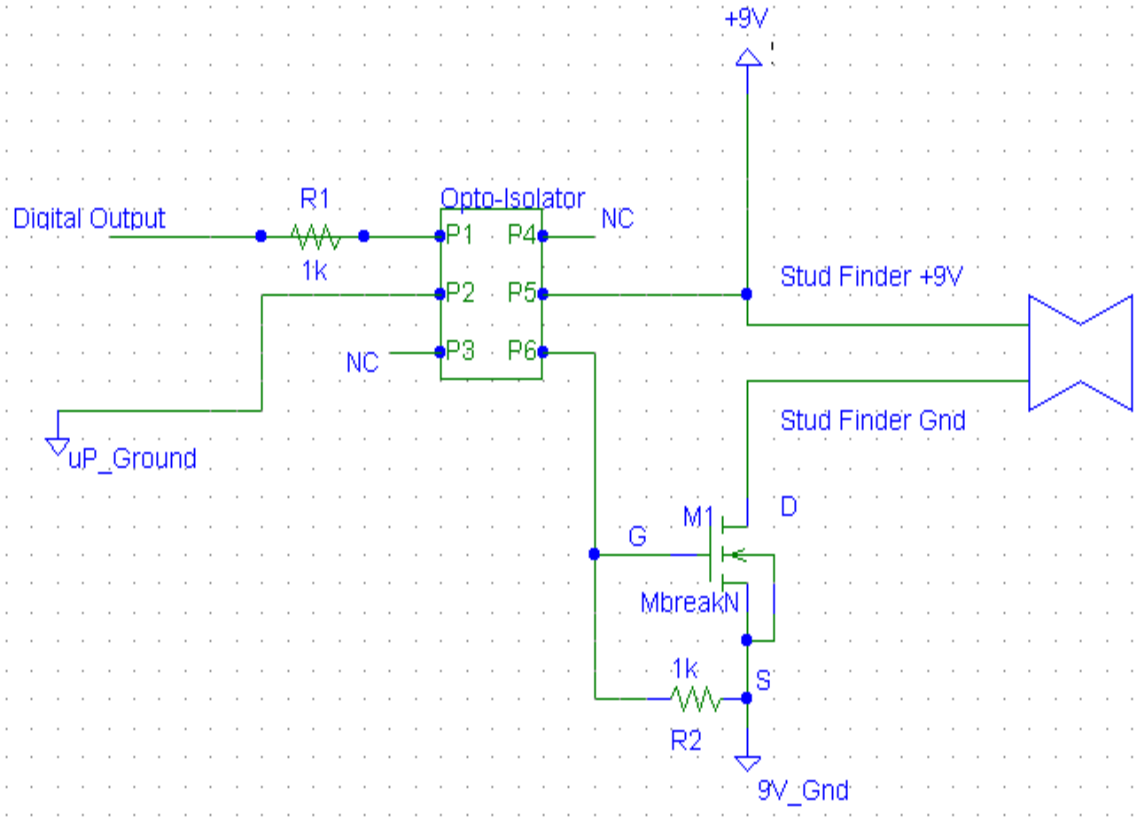
Battery Required: No

Battery Type: 9 Volt

(Source: www.homdepot.com, www.zircon.com)

The application of this sensor is the performance of its intended function. I have set up a circuit so that I can control it from my robot and am building a circuit to allow the stud detector to send its detection signal back to the robot.

The control circuitry allowing the the robot to turn the detector on and off is a MOSFET based circuit. The symbolic representation of this circuit follows:



The circuit that I use to control the solenoid is exactly the same, except for the fact that a different digital output is used to bring the signal in from the robot.

The circuit used to retrieve the signal from the stud finder was described above and only uses an opto-isolator like the one in the diagram. The digital input and the ground for the chip comes from a stud finder LED and the other side connects 5V to a microcontroller digital input.

The software algorithm for the stud finder is quite simple, the robot turns the stud detector on, wait for a second, turn it off, wait for a half second, turn it on again. This puts the stud detector in the "deep scan" mode which ensures the best results in finding studs. The robot then moves along the wall and whenever a high input signal comes in from the

stud detector, the robot plunges the solenoid.

Sensor data for the stud detector is simple. It is able to detect studs beneath sheetrock, whether the studs be wooden or metal. Also, on the walls in my apartment, the detector would find conduit. Which can be dangerous if you try to nail something to it. Graphs of this would be quite boring since in Normal scan mode, the detector found the studs 100% of the time and in deep scan mode it found the stud 100% of the time.

The most interesting lesson I have learned from the stud detector is a practical application of all of the MOSFET theory I learned in my Electronic Circuits class. I feel like there is at least a small practical application to all of that work and math.

CHAPTER 5

BEHAVIORS

In obstacle avoidance mode, the robot randomly traverses a room and avoids hitting things. Should it hit something, the robot makes a turn and continues.

In stud detection mode, the robot will start by ensuring that is it in a corner. A corner will be the operational requirement of the robot. This will ease the task of the robot finding where it is. Once the robot determines that it is in a corner (by detecting a wall with the side mounted IR and a bump with the rear sensor) the robot will turn on the stud detector and calibrate the IR values for wall following. It then moves along the wall. The wall following algorithm works as follows--The robot calibrates itself to the ambient light in the room by reading the IR values when the rear bump signals. The robot then begins to move. If the IR value from the front exceeds the calibrated one by 2 or more the robot turns back towards the wall. If the IR value from the front goes below the calibrated value by 2 or more, the robot turns away from the wall. The IR value from the rear IR ensures that the robot does not run into the wall. When it detects a stud, it will mark the floor by plunging the solenoid (making the right digital output go high), and move on. At the end of the wall it will be either stopped by the next wall in the corner, or it will turn towards the wall and stop if there is no corner and the wall simply ends.

CHAPTER 6

EXPERIMENTAL RESULTS

On demo day, November 29, 2001, Zircon performed its function as expected and quite well. I set it up next to two walls that I and my brother (Duane Ammons) built. Each wall consisted of a 2'x4' piece of drywall from Home Depot and a 2"x4"x8' pine board cut into 1'6" lengths, with one extra one measuring 2". These "studs" were screwed to the back of the drywall and stood as the wall Zircon scanned. We put down paper on demo day to make sure the robot didn't make a mess on the floor. After positioning the robot and turning it on, it ran along the wall and marked every stud in the wall. I was quite pleased.

CHAPTER 7

CONCLUSION

I set out in the beginning of this project to build a robot, and build a robot I did. Zircon, while not very sophisticated, performs behaviors based on the input he receives from his environment. These objectives satisfy the definition of an autonomous mobile agent (our fancy term for robot) and the requirements of this course. I have also managed to build something practical that has a possibility of seeing actual use in the field considering that my dad works as an electrician in the construction trade.

Some limitations of my design and implementation are hardware based, and others are software based.

The hardware based limitations rise from the fact that I kept the budget of the robot to just around $400.00. The equipment to control a robot available for that kind of money is not very sophisticated or accurate. The servos do not travel in a straight line when given the same pulse widths. The IR sensors fluctuate a bit and have a sweet spot distance, and if they get any closer, they no longer function effectively. The batteries would not last a full day of usage.

The software based limitations of the project rise from a lack of time and programming knowledge. Given more time, a more sophisticated wall following algorithm could be developed. A table of values could be used to move the robot at different speeds proportional to the distance away from the wall could be used to smooth out travel corrections.

I was quite pleased with the result of the project given that this was my first attempt at anything like this. For anyone else attempting a project like this, I would recommend that you not be scared of your equipment. I worked with people throughout the semester that were afraid to try things because they didn't know the result. My suggestion is that if you are going to burn circuits, and ruin microcontrollers this is the place to do it. Cut your teeth on technical mistakes here so that you don't make the same ones in industry where they can cost you your job.

In the future I hope to make some of the corrections I mentioned above and try to use the robot for my senior design project. I plan to add a little more sophisticated hardware and tweak the hardware already there. I am sure that I will get plenty of use out of this design at interviews and family functions to illustrate the fact that all my time at college didn't go to drinking and football.

APPENDIX

RELEVANT CODE

```
/*****************************SFAVOID.C*****************************/
/*This file adapted from obstacle avoidance code written by:Bret
Dennison*/
/*Adapted by: Wiley Ammons*/
/***************************** Includes
*********************************/

#include <tkbase.h>
/************************ End of includes
*******************************/
#define IR_THRESHOLD 117

void main(void)
/***************************** Main
*********************************/
{
  int rspeed, lspeed;

  init_analog();
  init_motortk();
  init_clocktk();
  init_serial();
  init_servos();

 wait(3000);

 while(1)
  {
    read_IR();

  if((IRDT[0] > IR_THRESHOLD ) || (IRDT[2] > IR_THRESHOLD ) )
    {
      if(IRDT[0] > IRDT[2])
      /* Start turning when something in front and keep turning until
nothing
        is in front */
        {
       rspeed = 4500;
       lspeed = 4500;
      }
      else
      {
        rspeed = 1500;
        lspeed = 1500;
          }

      servo(0,rspeed);
```

```
      servo(1,lspeed);
      wait(200);


      while((IRDT[0] > IR_THRESHOLD ) || (IRDT[2] > IR_THRESHOLD ) )
            read_IR();

    }
  else
    {
      lspeed = 1000;
      rspeed = 4000;
    }

      servo(0,rspeed);
      servo(1,lspeed);

    wait(35);
  }


}
/*************************** End of Main*****************************/
/*************************** END SFAVOID.C**************************/
```

```
/*********************************************************************
 * Title        studtst.c
 * Programmer     Wiley Ammons, Jr.
 * Adapted from Keith L. Doty
 * Date         November, 2001

 * Version  1
 *
 * Description
 *   This program allows you to test all of ZIRCON's basic features:
 *    IR Detectors
 *      Motors
 *      Servos
 *          Stud sensor
 * Select the appropriate test from the startup menu. When finished,
*  press reset on ZIRCON and select the next test.
 *
 * Use Hyperterm with the following settings:
 *
 *      COM1, 9600 Baud, 8 bits, No Parity, 1 Stop bit, No Flow
 *      of Control, VT100 Terminal, WrapLines.
 *
 *Usage: Verify functionality of ZIRCON's sensors.
 *********************************************************************/

/*************************** Includes*********************************/

#include <tkbase.h>
#include <stdio.h>

/************************ End of includes*****************************/

/*************************Prototypes*********************************/
void sensors_tst(void);
void servo_tst(void);
void finder_tst(void);
void show_stud(void);
void sol_tst(void);
void suite_tst(void);

/************************ End of Prototypes**************************/


/*************************Constants*********************************/

/*IR emitter output port driver, the output latch at address 0xffb9 */
#define IRE_OUT   *(unsigned char *)(0xffb9)

/*Constant for driving all the 40KHz modulated IR emitters on when
loaded into IRE_OUT */
#define IRE_ALL_ON 0xff
```

```
/*Constant for turning all the 40KHz modulated IR emitters off when
loaded into IRE_OUT */
#define IRE_ALL_OFF 0x00

/************************ End of Constants*****************************/

void main(void)
/****************************** Main**********************************/
{
  unsigned int bl, cv, fb, rb, run_test;
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* clear screen */
  char place[]= "\x1b[1;1H";        /* Home cursor */

  init_analog();
  init_clocktk();
  init_serial();
  init_servos();

  IRE_OUT=0x00;
  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/

  printf("\tTitle\tStudtst.c\n"
  "\tProgrammer\tWiley Ammons, Jr.\n"
  "\tAdapted from\tKeith L. Doty\n"
  "\tDate\t\tNovember 25, 2001\n"
  "\tVersion\t\t1\n\n");

  printf("Select:\n"
         "   1. Test ZIRCON Sensors \n"
           "   2. Test ZIRCON Servos\n"
           "   3. Test ZIRCON Finder\n"
             "   4. Test ZIRCON Solenoid\n\n");
  run_test= read_int();

    switch (run_test)/*Select a test procedure */
       {
         case 1: sensors_tst(); break;
         case 2: servo_tst(); break;
         case 3: finder_tst(); break;
         case 4: sol_tst(); break;
       }



}
/*************************** End of Main******************************/


void sensors_tst(void)
```

```
/*************************** sensor_tst ******************************
 * Description
 * Displays ZIRCON's sensor inputs, 2 IR detectors,
 * and rear bumper.
 *
 *Usage: Verify functionality of sensors.
 ********************************************************************
*/
{
  int rb;
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* clear screen */
  char place[]= "\x1b[1;1H";        /* Home cursor */

  printf("%s", clear);  /*clear screen*/
  printf("%s", place);  /*home cursor*/

      printf("\x1b[2;1H Front  IR:");
      printf("\x1b[4;1H Rear   IR:");
      printf("\x1b[6;1H Rear Bump:");

while(1)  /* Update the sensor fields indefinitely */
  {

  read_IR();     /* Read all IR sensors */

  rb = rear_bumper();         /*Returns rear bumper value*/

      printf("\x1b[2;12H %u", IRDT[0]);  /*select IRDFM*/
    printf("\x1b[4;12H %u", IRDT[2]);
    printf("\x1b[6;12H %u", rb);

  } /*end while*/


}

/*************************** sensors_tst*****************************/


void servo_tst(void)
/*************************** servo_tst ******************************
 *
 * Description
 *
 *   Type in the pulse width (1000-5000) to send to the servos. If the
 *   servos are operating properly, the robot will spin clockwise or
 *   counter-clockwise depending on the widths sent to them.

 ********************************************************************/
{
int i, rb, servo_speed;
```

```
/* VT100 clear screen */
  char clear[]= "\x1b\x5B\x32\x4A\x04";

/* VT100 position cursor at (x,y) = (3,12) command is "\x1b[3;12H"*/
  char place[]= "\x1b[1;1H";  /*Home*/

  printf("%s", clear);
  printf("%s", place);

    while(1)
  {

  printf("Enter Servo Pulse width = \n\n");
  servo_speed = read_int();

  servo(0,servo_speed);
  servo(1,servo_speed);
  }

}
/*************************** servo_tst ******************************/


void finder_tst(void)
/*************************** finder_tst*****************************
 * Description
                                      *
 *   Twiddles the bits needed to run the studfinder circuitry and to
get*
 *   the answer returned.
      *
 ***********************************************************************/
{
  unsigned int find_test;
  int i;

/* VT100 clear screen */
  char clear[]= "\x1b\x5B\x32\x4A\x04";

/* VT100 position cursor at (x,y) = (3,12) command is "\x1b[3;12H"*/
  char place[]= "\x1b[1;1H";  /*Home*/

  printf("%s", clear);
  printf("%s", place);

  wait(500);

  while(1)
  {
  printf("%s", clear);
```

```
  printf("%s", place);

  printf("Select:\n"
         "   1. Turn Finder on\n"
           "   2. Turn Finder off\n"
           "   3. Display Finder Status\n"
             "   4. Test Special Sensor Suite\n\n");

    find_test= read_int();

    switch (find_test)/*Select a test procedure */
       {
         case 1: IRE_OUT=0x20; break;
         case 2: IRE_OUT=0x00; break;
         case 3: show_stud(); break;
         case 4: suite_tst(); break;
         default: break;
       }
  }

}
/*************************** finder_tst ******************************/


/*************************** show_stud  ******************************/

void show_stud(void)
/*********************************************************************
 *    This subroutine gets the data back from the studfinder, and
       *
 *      displays it to the terminal
                    *
 *********************************************************************/

{
  unsigned int stud_flag;
  int i;

/* VT100 clear screen */
  char clear[]= "\x1b\x5B\x32\x4A\x04";

/* VT100 position cursor at (x,y) = (3,12) command is "\x1b[3;12H"*/
  char place[]= "\x1b[1;1H";  /*Home*/

  printf("%s", clear);
  printf("%s", place);

  while(1)
  {
  read_IR();     /* Read all 14 IR sensors */
  printf("\x1b[3;1H Stud = \b%u", IRDT[5]);
```

```
  }
}
/*************************** show_stud ******************************/


void sol_tst(void)
/*********************************************************************
 *    This subroutine tests the solenoid by giving you an option to
*
 *      plunge it or allow it to retract
              *
 *********************************************************************/

{
  unsigned int noid_test;
  int i;

/* VT100 clear screen */
  char clear[]= "\x1b\x5B\x32\x4A\x04";

/* VT100 position cursor at (x,y) = (3,12) command is "\x1b[3;12H"*/
  char place[]= "\x1b[1;1H";  /*Home*/

  printf("%s", clear);
  printf("%s", place);

  wait(500);

  while(1)
  {
  printf("%s", clear);
  printf("%s", place);

  printf("Select:\n"
         "   1. Turn Solenoid on\n"
           "   2. Turn Solenoid off\n\n");

  noid_test= read_int();

    switch (noid_test)/*Select a test procedure */
       {
         case 1: IRE_OUT=0x40; break;
         case 2: IRE_OUT=0x00; break;
       }
  }
}
/*************************** sol_tst ******************************/

void suite_tst(void)
{
      IRE_OUT=0x20;
```

```
      while(1)
      {
       read_IR();

       if(IRDT[5] > 100)
            IRE_OUT=0x60;
       else
            IRE_OUT=0x20;
      }
}
/************************* END sol_tst  ****************************/
/************************* END studtest.c ***************************/
```

```
/***************************Zirconf.c*********************************/
/*This is the final behaivior code for Zircon. This is the code used on
 * demo day. */
/*************************** Includes*********************************/

#include <tkbase.h>

/************************ End of includes*****************************/
#define IR_THRESHOLD 2

/*IR emitter output port driver, the output latch at address 0xffb9 */
#define IRE_OUT   *(unsigned char *)(0xffb9)

void main(void)
/***************************** Main**********************************/
{
  int rspeed, lspeed, rcalib, fcalib, rb;
  char clear[]= "\x1b\x5B\x32\x4A\x04";  /* clear screen */
  char place[]= "\x1b[1;1H";        /* Home cursor */

  init_analog();
  init_clocktk();
  init_serial();
  init_servos();

  rb = 0;

while (rb < 100)
        rb = rear_bumper();

  IRE_OUT = 0x20;

  wait(750);

  IRE_OUT = 0x00;

  wait(750);

  IRE_OUT = 0x20;

  wait(750);

      read_IR();
      rcalib=IRDT[2];
      fcalib=IRDT[0];

 while(1)
  {
    read_IR();

      if(IRDT[5] > 100)
```

```
            IRE_OUT=0x60;
       else
            IRE_OUT=0x20;

      rspeed=3100;
      lspeed=2850;

  if (IRDT[0] > (fcalib + IR_THRESHOLD))/*going towards wall front*/
    {/*turn away*/
        rspeed = 3200;
        lspeed = 3000;

      servo(0,rspeed);
      servo(1,lspeed);
      }

  while((IRDT[0] > (fcalib + IR_THRESHOLD)) || (IRDT[2] < (rcalib -
IR_THRESHOLD)))
      {
       read_IR();
             if(IRDT[5] > 100)
                  IRE_OUT=0x60;
             else
                  IRE_OUT=0x20;
            }


      rspeed=3100;
      lspeed=2850;

        servo(0,rspeed);
      servo(1,lspeed);
        wait(100);

  if (IRDT[0] < (fcalib - IR_THRESHOLD))/*front going away*/
    {/*turn towards*/
      lspeed = 3000;
      rspeed = 2800;

      servo(0,rspeed);
      servo(1,lspeed);
    }

  while((IRDT[0] < (fcalib - IR_THRESHOLD)) || (IRDT[2] > (rcalib +
IR_THRESHOLD)))
                  {
                   read_IR();
                   if(IRDT[5] > 100)
                        IRE_OUT=0x60;
                   else
                        IRE_OUT=0x20;
```

```
                }


      rspeed=3100;
      lspeed=2850;

      servo(0,rspeed);
      servo(1,lspeed);
        wait(100);

  }


}
/*************************** End of Main*****************************/

/*************************** END zirconf.c***************************/
```