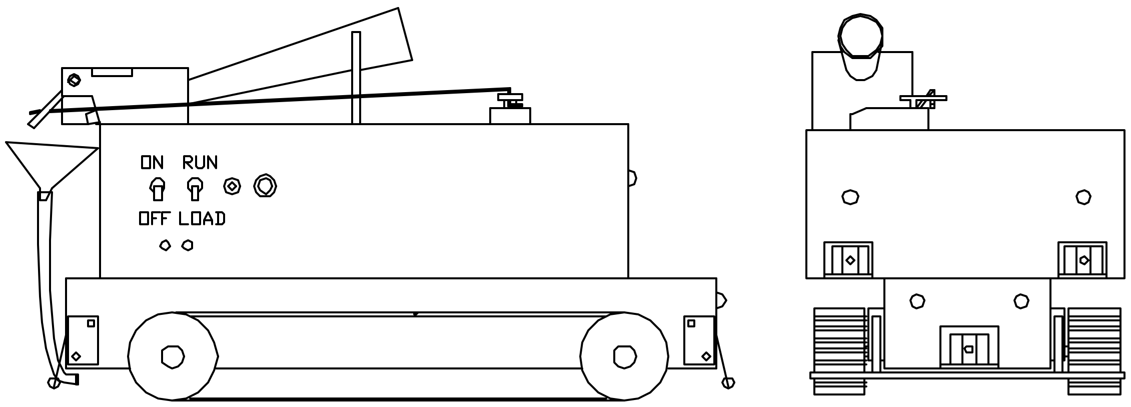
University of Florida
Department of Electrical and Computer Engineering
EEL5666
Intelligent Machines Design Laboratory

Magnetic Mine Hunter (MMH) Final Report



Name: Lee Cofer
Date: 1/22/02
TAs: Aamir Qaiyumi
Uriel Rodriguez
Instructor: A.A. Arroyo

Table of Contents

Abstract

Magnetic Mine Hunter (MMH) is an autonomous tracked vehicle that randomly moves around on the floor within an area outlined by black tape. While wondering around it is searching for magnets on the floor that represent mines. When a magnet is encountered MMH will mark the magnet by dropping a metal nut at the location of the magnet. While searching for the magnets MMH will avoid running into objects.

Executive Summary

Magnetic Mine Hunter (MMH) is an autonomous robot that simulates mine hunting. A black line is used to outline the area designated as the minefield to be searched. MMH randomly searches the minefield for magnets on the ground, which represent the mines. When a magnet is found MMH stops and marks the location by dropping a metal BB in close proximity to the magnet. MMH then continuous searching the field. While searching MMH avoids obstacles by going around the obstacles that lye in MMH's path. As a backup to the obstacle avoidance MMH is equipped with front and rear bumpers so that when an obstacle is hit MMH will stop, back away, then change its path.

When MMH is first placed within the mine field with the power turned on MMH waits to start its search process until the front bumper is pressed. Once the bumper has been pressed MMH will remain still while the light sensitive CdS cells used to detect the border are calibrated. When the calibration is complete MMH starts moving forward while reading the sensors to detect the magnets, the border, and obstacles. If a border is detected by the CdS cells MMH will backup and randomly pick a new path to continue on. When an obstacle is detected using IR emitters and IR detectors, MMH will steer to the right or the left of the obstacle. And when the Hall Effect sensors on the underside of MMH detect a magnet, MMH will stop and mark the magnets location with a BB.

MMH will continuously search a minefield until it is turned off. If given sufficient time the entirety of the field will be searched and all of the mines will be detected. The amount of time that it would take to search an entire field is unpredictable since MMH searches randomly.

Introduction

Soldiers in the United States Armed Forces have to be cautious of mines in hostile territories. Mines can be found buried underground and in surf zones off of coastal lands. Mines are an extreme hazard to soldiers and the vehicles used to insert soldiers to the battlefield. Searching for mines is a very dangerous job, but it is necessary for the safety of our troops. In the past, detecting mines was very dangerous because soldiers would have to be sent out to find the mines. This hazard is no longer necessary with the technology we have today. Soldiers can now be replaced with machines that can find the mines so that a life is not being risked. Magnetic Mine Hunter (MMH) is an early stage of an autonomous agent that could one day be used to save the lives of our troops. MMH is designed to randomly search a designated area for magnets that represent mines. While searching for mines MMH will avoid obstacles that obstruct its path. When a mine is found MMH will mark the mine by dropping a metal BB near the location of the mine. In this paper I will propose how MMH was built, how works, and the sensors that are used to achieve the desired behaviors.

Integrated System

A diagram of the Integrated system can be seen in Figure 1. The brain of MMH is the MRC11 microprocessor board and the MRSX01 sensor expansion board, which are both made by Mekatronix. A picture of the boards can be seen in Figure 2. These two boards combined control all of the components of MMH. The components include two DC motors (hacked servos), one standard servo, three IR sensors, four bumper switches, two CdS cells, and seven Hall Effect sensors.

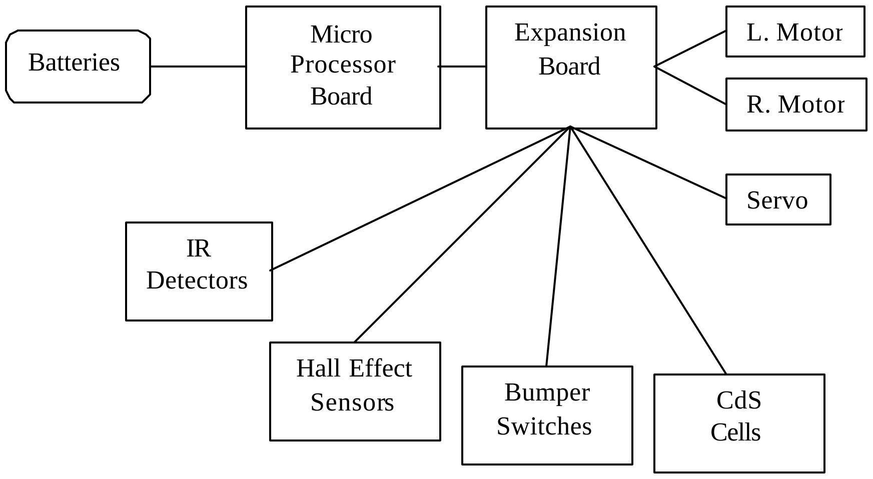


Fig. 1 Complete System

The two DC motors are used for movement of the robot so that MMH can move forward, reverse, turn left, and turn right. The motors are hacked Futaba S3003 servos. Hacked means that all of the electronics were removed from the servo and only the motor, gears, and the housing of the servo are left. These motors are controlled using the motor driver IC on the MRSX01 expansion board. A standard Futaba S3003 servo is used to control the door of the mechanism that drops the BBs onto the magnets. Three Sharp Can IR detectors and four IR LED emitters are used for the obstacle avoidance. There are two bumper switches on the front and back of the robot used to detect when the obstacle avoidance fails and MMH runs into an obstacle. Two CdS cells and two green LEDs are used for border detection. The CdS cells allow MMH to detect the amount of light that is being reflected off the floor. So when MMH drives over the black border less light is reflected than when on the lighter search area thus letting MMH detect that it is driving out of the search area. Seven Hall Effect sensors on the bottom of MMH are used to detect the magnetic field produced by the magnets. So the Hall Effect sensors make it possible to perform magnet/mine detection.

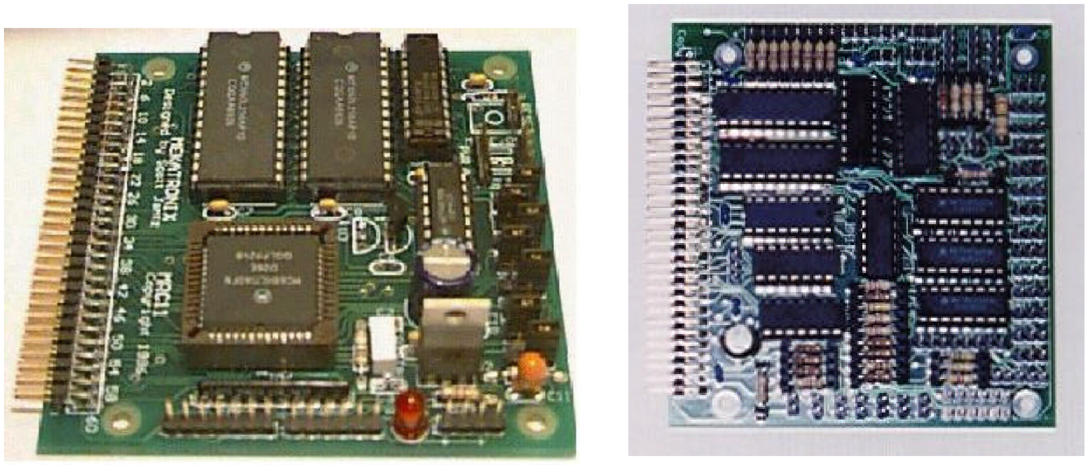


Fig. 2 MRC11 (left) and MRSX01 (right)

Mobile Platform

The mobile platform for MMH shown in Figure 3 is constructed out of 1/8" plywood. The design was cut out on the IMDL T-tech manchine. The overall dimensions of the platform are 10" long by 6" wide by 5" tall. The platform is a tank style that houses all of the electronics. The tank style was chosen for its durability and because most of the electronics are enclosed from outside elements. The platform consists of two compartments. The top compartment is a rectangular box that is approximately 8" x 6" x 3". The top compartment houses the processor and expansion boards, two of the IR emitters, two of the IR detectors, on/off switch, run/load switch, reset button, charge jack, and the three feedback LEDs. The bottom compartment is 10" x 3" x 2". The bottom compartment houses the other IR detector, two IR emitters, seven Hall Effect sensors, four bump switches, two CdS cells and LEDs, the battery pack, and the drive motors. The top cover of the platform is removable so that the systems on the inside are accessible. Mounted on the top cover is the BB dropper mechanism so that the BBs can be dropped off the back of the platform then directed towards the location of the detected mine. The main function of the platform is to have an enclosure in which to mount and protect the electronics.

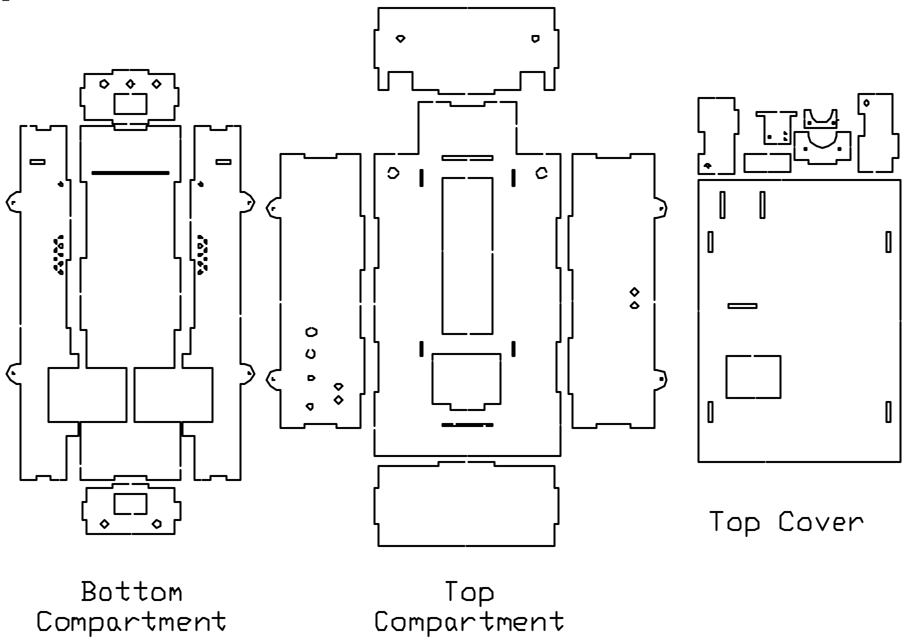


Fig. 3 Platform

Actuation

MMH requires two DC motors for mobility and a servo to control the BB dropping mechanism that is used to mark the magnets by dropping a BB on them. The purpose, theory, and application of the motors and servos are as follows:

1.) Drive Motors

Purpose:

The two drive motors will be used to control the movement of the vehicle.

Theory:

The drive motors require H-bridges to control the direction of the motors so that a mechanical means of switching the direction of the current to the motors is not required. The motors also need gears so that the speed of the motors can be slowed down and produce more torque. So the power from the motors is sent to the gears and from the gears to wheels that move the vehicle.

Application:

The drive motors are hacked servos from [2], which means the electronics of the servo were removed leaving the motor and gears inside the servo casing. The servos were also modified so that they can rotate 360°. The hacked servos are mounted on the bottom back of the vehicle. Two wheels are attached to the servos and two free turning wheels are attached to the front of the vehicle. Tank style tracks are attached between the front and rear wheels on each side. The tracks are individually controlled so the vehicle can move forward, backwards, and turn in place.

2.) BB Dropper Control Servo

Purpose:

The servo controls the door of the BB dropper mechanism that allows a single BB to drop on the magnets/mines to mark them.

Theory:

A servo is a circuit and motor that is designed to be precisely controlled over an angle range that is usually around 90 degrees. The precise control is required in this application since the door on the BB dropper mechanism must only be opened far enough and for proper amount of time so that only a single BB is dropped.

Application: BB Dropper Mechanism

The device used to mark the magnets by dropping BBs consists of tubular funnel, a door to keep the BBs from all falling out, a mount to hold up the funnel and door, and a servo and linkage to control the door. In order to slow down the rate that the BBs fall out of the funnel the funnel is mounted at an angle of approximately 30 degrees above level. The door was mounted in front of the funnel so that when it is pulled back no BBs can escape and when pushed forward the BBs have enough room to flow out of the funnel. The servo was placed behind the funnel with a servo horn mounted on top in order to have a connection for the linkage. The linkage is a small metal rod that connects the servo to the door. When the servo turns counter clockwise the door opens and when it turns clockwise the door will shut. The construction of the BB dropping mechanism can be seen in Fig. 4 as it would be mounted on top of the MMH platform.

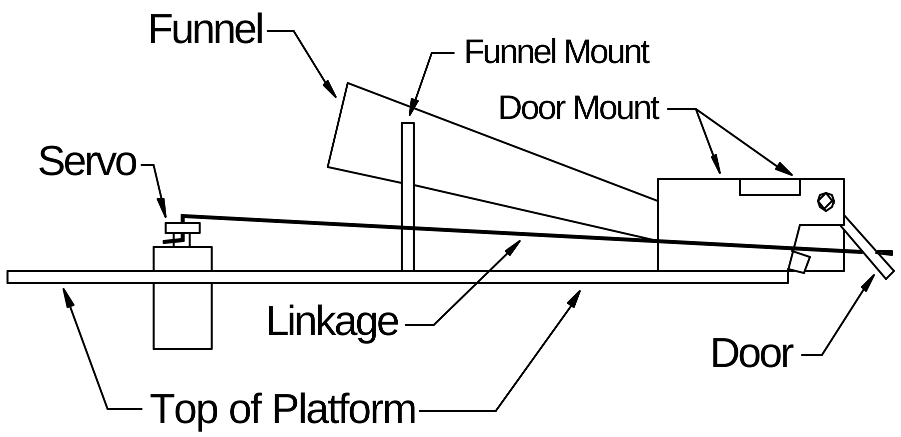


Fig. 4 BB Dropper Mechanism

Sensors

MMH uses several types of sensors in order to complete the behaviors and tasks that are demanded from it. The sensor suite for MMH includes IR sensors for obstacle avoidance, bumper switches for object detection, Cadmium Sulfide Cells for detecting the perimeter of the search area, and Hall-Effect sensors for detecting the magnets that represent mines. The purpose, theory, and application of these sensors can be found below:

1.) Bump Switches

Purpose:

The bump switches are used to detect when MMH runs into an obstacle. The bump switches are used as backup for the obstacle avoidance system.

Theory:

The bump switches act like a normal switch in that when not triggered it is in the open position and when triggered it creates a short circuit. Pressing the lever on the front of the housing triggers the switch. The switches are used when the obstacle avoidance sensors fail to detect an obstacle in the vehicles path. When the vehicle runs into an obstacle the switches will be triggered letting the software now that it can no longer continue on its current path.

Application:

Two bump switches are placed on the front of the vehicle and two on the rear. A cross member attaches the two switches together so that the entire front and rear of the vehicle will be covered by the bump switches. When the vehicle runs into an obstacle these switches will be activated letting the hardware and software know that MMH can no longer continue on its present path. MMH will then back up and randomly choose a new path.

2.) IR Emitters and Detectors

Purpose:

The IR emitter and detector pairs are part of the obstacle avoidance sensor suite.

Theory:

The IR emitter transmits IR away from the vehicle at a frequency of 40kHz. The signal must be 40kHz because that is the only frequency that the IR detectors being used can detect. If the IR hits an object it will bounce back to the vehicle and the IR detector will detect the IR. The closer an obstacle is to the vehicle the stronger the IR signal will be when it returns to the vehicle, thus the output of the detectors will be higher. The code for the IR detectors will be written such that when the output of the detector reaches a set threshold the path of the vehicle will be adjusted appropriately.

Application:

The IR detectors being used have a digital output, but for obstacle avoidance an analog output is needed. An analog hack from [1] must be performed on the detectors so that they will output an analog signal. When the sensors have been hacked the emitter/detector pairs will be placed on the vehicle so that they are facing away from the vehicle parallel to the ground. The three detectors are all facing forward. Two of the IR emitters are pointed forward and the other two are directed in a cross view so that the entire area in front of the vehicle is covered by IR.

3.) Cadmium Sulfide Cells

Purpose:

The Cadmium Sulfide Cells are used to detect the black tape on the floor that outlines the designated area to be searched.

Theory:

Cadmium Sulfide Cells are light sensitive sensors that act as variable sensors. When the cells detect light the resistance of the sensor decreases.

If the CdS cell is used as one of the resistors of a voltage divider circuit the output of the voltage divider can be measured to determine the amount of light that is being reflected onto the cell. See figure 5 for the voltage divider circuit. Since darker colors reflect less light than lighter colors the cells can distinguish between contrasting colors

.

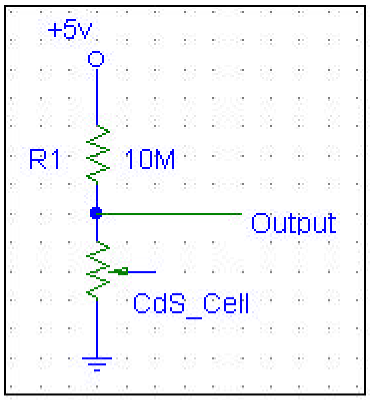


Fig. 5 CdS Cell Voltage Divider

Application:

In order for the CdS cells to detect light from a specific location the sensor was enclosed in a heat shrink tube with the back end closed off, which was discovered from Aamir Qaiyumi. The front of the tube was left open so that light can be picked up from the direction the tube is pointed in. The Cadmium Sulfide Cells are mounted inside the front left and right of the platform and sticking out of the bottom. The cells will be directed towards the floor so that the sensors can detect when they are above the black tape. Two green LEDs are also mounted beside the CdS cells directed towards the floor for illumination. The LEDs illuminate the floor underneath the CdS cells so that the sensors work independently from exterior lighting.

When MMH is first turned on the CdS cells are calibrated to the amount of light reflected from the floor. The calibration allows MMH to be used on different color surfaces as long as the color of the surface is lighter than the black tape used to designate the perimeter of the search area. When MMH drives over the tape the MMH will back up then randomly choose a new path so that the designated area is not left.

4.) Hall-Effect Sensors

Purpose:

The Hall-Effect sensors are used to detect the magnets on the floor that represent mines.

Theory:

Hall-Effect sensors are analog sensors used to detect magnetic fields. When a magnetic field is placed close to the sensor the output represents the strength of the field.

Application:

Seven Hall-Effect sensors are mounted on the front bottom of the vehicle. The faces of the sensors are parallel with the floor so that when the vehicle drives over a magnet the sensor can detect it.

Vendor:

Digi-Key
Part #: DN6848

Behaviors

Obstacle Avoidance:

MMH implements obstacle avoidance using IR emitters and detectors. When an obstacle is in front of MMH IR is reflected off the object back to the IR detectors. If the strength of the IR is larger than a set threshold value MMH will turn to avoid the obstacle. If the right or center IR detector receives the highest value above the threshold value MMH will turn to the left. If the left IR detector receives the highest value MMH will turn to the right. If the obstacle avoidance system fails and the bump switches are triggered MMH will stop, back up, and then randomly choose a new path.

Mine Detection and Marking:

When the Hall Effect sensors detect a mine MMH will immediately stop, drop a single BB from the BB Dropper mechanism onto or near the mine. MMH will then start moving forward for 300ms with the Hall Effect sensors off. After the 300ms the Hall Effect sensors are out of range from the mine so that the sensors can be turned back on without detecting the same mine.

One lesson learned when working on the mine detection is when testing Reed Switches for use as the sensor to detect the magnetic field of the magnet. It was discovered that the Reed Switches used required very strong magnets to trigger them than the magnets I decided to use. Also the Reed Switches use metal reeds, which cause the magnets to attach themselves to the sensor.

Another lesson learned was that the BBs when dropped from the BB dropper mechanism they would often bounce off away from the magnet. In order to get the BBs to attach themselves to the magnet, a tube was added to the BB dropper that directed the BBs to the magnets.

Border Detection:

MMH performs border detection using two CdS cells and two green LEDs. When MMH is first turned on the CdS cells are calibrated then mine searching begins. When MMH drives over the border of the search area MMH will stop, back up, and then randomly choose a new path.

Experimental Layout and Results

Experiment 1: Hall Effect Sensor Sensitivity

This experiment was designed to test the sensitivity of the Hall Effect sensors. The Hall Effect sensor used is a Panasonic DN6848, relative data can be found in Appendix B. This sensor was chosen because it is one of the only unipolar Hall Effect sensors found in a three prong package. The three prong package was desired so that it could be easily connected to the IRDT ports on the MRSX01 expansion board. In order to test the sensitivity of the sensor a round magnet with a diameter of ¾" was placed flat on a table with the south pole of the magnet facing up. It is essential for the south pole of the magnet to face the sensor since the sensor is unipolar and can only detect the south pole. Code was written to continuously display the output of the sensor on a computer screen. The code used can be found in Appendix A. The sensor was then placed 1" above the center of the magnet and the output was recorded. The sensor was then lowered towards the magnet at 1/16" intervals, until the sensor was sitting directly on top of the magnet. The magnet was then brought back up to 1" above the magnet, but this time it was offset from the center of the magnet by 1/16". With the 1/16" offset the sensor was again lowered towards the magnet at 1/16" intervals. The procedure of lowering the sensor towards the magnet was then repeated with the sensor offset from the center of the magnet at intervals of 1/16", until the magnet was offset by ½".

Table 1 below contains the results of Experiment 1 and Chart 3 follows with a graph of Distance above Magnet vs. Analog Output of the Hall Effect sensor. The table shows that the sensor detects the magnet as long as it is within 5/16" from the center of the magnet. Since the magnet has a diameter of approximately ¾" this means that the sensor must be directly over the magnet to detect it. So these results show that the Hall Effect sensor will detect a magnet as long as it is above the magnet and it is no more than 3/8" from the magnet.

Hall Effect Sensor Sensitivity

Offset from Center of Magnet (inches)

Distance from Top of Magnet (inches)

	0	1/16	1/8	3/16	1/4	5/16	3/8	7/16	1/2
0	1	1	1	1	1	1	38	44	50
1/16	1	1	1	1	1	1	34	40	47
1/8	1	1	1	1	1	1	1	37	48
3/16	1	1	1	1	1	1	1	46	49
1/4	1	1	1	1	1	1	35	47	50
5/16	1	1	1	1	1	1	42	48	49
3/8	1	1	1	1	1	36	44	50	51
7/16	1	37	45	46	44	38	44	49	52
1/2	45	45	44	42	48	43	47	50	54
1	53	46	47	54	50	51	52	51	53

* Results are from the HC11's A/D converter.

** 1 = magnet detected; greater than 1 = magnet not detected

Table 1

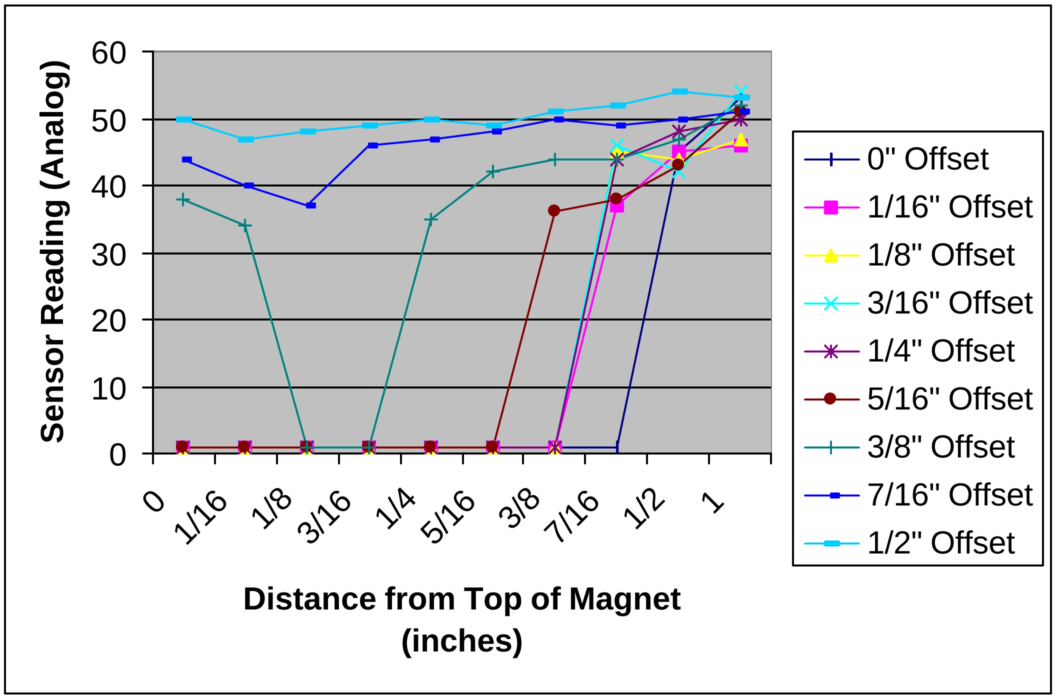


Chart 1

Experiment 2: BB Dropper

This experiment was designed to find the range of movement needed for the door and how long the door needs to be opened in order for the BB Dropper mechanism to only drop one BB at a time. The movement of the door is controlled by a servo through a linkage arm. Test code (see Appendix A) was written so that the servo's position could be controlled. It was found that servo had a range of movement of approximately 180 degrees that is mapped to the integer position values between 500 and 4000. Once the mechanism was built the first task was to find the door to be open and closed. When the door is shut it needs to be flush with the opening at the front of the funnel so that no BBs can fall out. The position of the servo was adjusted until the door became flush with the funnel's opening. This closed position was found at the servo position value of 2200. The open position needed to leave enough room for the BB to fall out of the funnel. The space between the funnel and the door needs to be slightly larger than the size of the BB, so that the BB does not get caught between the funnel and the door. The amount of extra space required was found to be approximately 1/16". This space was obtained at the servo position value of 2800.

Now that the proper open and closed positions of the door have been found, the amount of time that the door needs to remain open to drop only one BB must be discovered. This experiment was performed by opening the door for a certain amount of time and counting the number of BBs that fell out of the funnel. The door was left open for 10 different lengths of time, which ranged from 60ms to 150ms at 10ms intervals. The test was performed three times for each of the 10 lengths of time tested to ensure more reliable results. This experiment was performed twice once with the funnel full of BBs and a second time with only 10 BBs in the funnel. The reason for performing this experiment with a full funnel and an almost empty funnel was to ensure that the amount of time selected would perform well in both circumstances.

Table 2 below contains the results of Experiment 2. When performing the tests sometimes a BB would get caught between the door and the funnel. In these instances the trapped BB would be counted as half of a BB, thus accounting for the fraction of a BB falling as seen in the table. As highlighted in the table, the only time that one BB dropped out of the funnel for each of the trials performed for the full funnel and almost empty funnel was at 80ms. So as the mechanism is set up now, an 80ms time lapse between opening the door and closing it gives the best chance of only one BB falling from the funnel.

Full Funnel				
Time Open (msec)	Trial 1 (BBs)	Trial 2 (BBs)	Trial 3 (BBs)	Ave. (BBs)
60	0	0	0	0.00
70	1	1	0	0.67
80	**1**	**1**	**1**	**1.00**
90	1	1.5	1	1.17
100	1	0.5	1	0.83
110	1	2	1	1.33
120	1	1	2	1.33
130	1.5	1.5	1.5	1.50
140	1.5	2	2	1.83
150	2	2.5	3	2.50

10 BBs in Funnel				
Time Open (msec)	Trial 1 (BBs)	Trial 2 (BBs)	Trial 3 (BBs)	Ave. (BBs)
60	0	0	0	0.00
70	1	0	0	0.33
80	**1**	**1**	**1**	**1.00**
90	1	1	1	1.00
100	1	1	1	1.00
110	1	1	1	1.00
120	1	1	2	1.33
130	1	1	1	1.00
140	1	2	2	1.67
150	2	2	2	2.00

Table 2

Experiment 3: IR Threshold

This experiment was necessary to find the threshold value for the IR detectors. The code irtest.c found in appendix A was used to display the values taken from the IR detectors. This experiment was performed with the IR detectors and IR emitters mounted in the platform. The readings from the IR detectors were read when different sizes of obstacles were placed at several positions in front of the platform. It was determined that the threshold value for the IR detectors should be 100. 100 is the value read when obstacles were approximately 1.5ft in front of the platform.

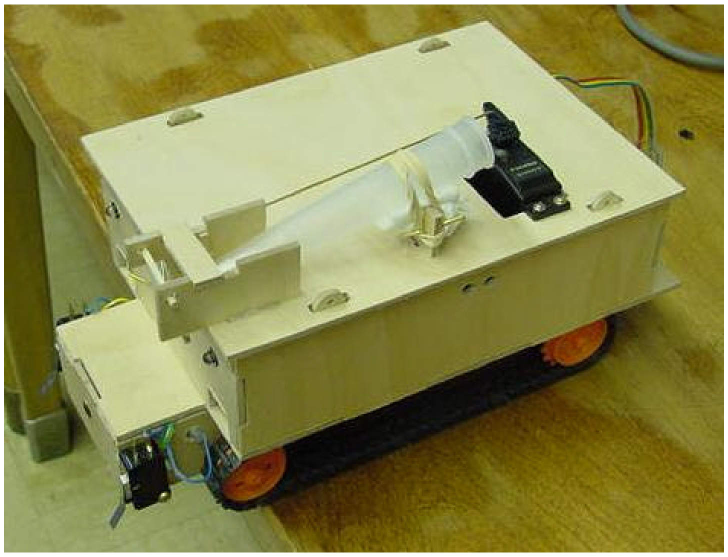
Experiment 4: CdS Cell Threshold

This experiment was necessary to find the threshold value for the CdS cells. The code Cdstest.c found in appendix A was used to display the values taken from the CdS cells. This experiment was performed with the CdS cells and green LEDs mounted in the platform. The readings from the CdS cells were read when pieces of paper of several colors were placed under the CdS cells. It was determined that the threshold value for the CdS cells should be 70% of the value found when MMH is on the area to be searched.

Conclusion

MMH is now a fully autonomous magnetic mine hunter, which randomly searches for mines and marks them in a designated area while avoiding obstacles. The one improvement that I would like to make in the current design of MMH is the mine marking device. The BBs do not always attach themselves to the magnet. If I had more time I would redisign the BB dropper so that the BB would be dropped directly on top of the magnet.

The limitation of MMH as a mine hunter is that it only simulates mine hunting of magnets on a flat surface. Future work to be done is to redesign the platform so that the rugged terrain of the outside can be searched. Sonar also needs to be added to the sensor suite since IR does not work well in sunlight. Also more sensors are needed to detect real mines. Some possibilities could be a metal detector and a sensor that can smell explosives.



<u>**Documentation**</u>

Thanks to
IMDL class: Instruction from Dr. Arroyo, Aamir Qaiyumi, Uriel Rodriguez, and Dr. Schwartz.

[1] Keith L. Doty, Electrical Engineering Department, Univ. of Florida, Gainesville, FL, <u>Sharp IR Sensor Hack for Analog Distance Measurement</u>, 1996.

[2] Keith L. Doty, Electrical Engineering Department, Univ. of Florida, Gainesville, FL, <u>MS SERVO HACK</u>.

[3] Keith L. Doty, Electrical Engineering Department, Univ. of Florida, Gainesville, FL, <u>MRC11 Assembly Manual</u>; Mekatronix 1999.

[4] Keith L. Doty, Electrical Engineering Department, Univ. of Florida, Gainesville, FL, <u>MRSX01 Assembly Manual</u>; Mekatronix 1999.

Vendors

Digi-key
701 Brooks Avenue South
P.O. Box 677
Thief River Falls, MN 56701-0677
1-800-344-4539
www.digikey.com

Mekatronix
316 NW 17 Street Suite A
Gainesville, FL 32603
352-376-7373
www.mekatronix.com

Radio Shack
3315 SW Archer Road
Gainesville, FL 32608
352-375-2426

Mondo-tronics, Inc.
4286 Redwood Hwy PMB-N
San Rafael, CA 94903
415-491-4600
www.RobotStore.com

Appendices

Final Code:

```
/***************************************************************************
    * Title        integra9.c                          *
    * Programmer        Lee Cofer                                        *
    * Date        4/18/2002                          *
    * Version 9                                              *
    *                                                                *
    * Description                                                         *
    *   This program performs obstacle avoidance, magnet detection,      *
    *                       magnet marking, and border avoidance.              *
    ***************************************************************************/
    /*************************** Includes **********************************/
    #include <tkbase.h>
    #include <servotk.h>
    #include <hc11.h>
    #include <mil.h>
    #include <vectors.h>
    /************************* End of Includes *****************************/

    /*************************** Constants **********************************/
    #define CDSL CDS[0]
    #define CDSR CDS[5]
    #define IRE_OUT   *(unsigned char *)(0xffb9) /* Address of Digital_Out */
    #define IRE_ON 0x1f  /* IR emitters on */
    #define IRE_OFF 0x00 /* IR emitters off */
    #define LED_AVOID 0x3f /* Avoid LED on */
    #define LED_MINE_DET 0x5f /* Mine detection LED on */
    #define LED_BORDER 0x9f /* Border LED on */
    #define SERVO1 0
    #define IRR analog(2)
    #define IRL analog(4)
    #define IRC analog(5)
    #define IR_THRESHOLD 100
    #define    CDS_THRESHOLD 95
    #define FORWARD 100
    #define REVERSE -100
    #define HFORW 80
    #define HREV -80
    #define STOP 0
    #define RIGHT_MOTOR 1
    #define LEFT_MOTOR 0
    #define BUMPER_FUZZY_ZERO 12 /* Noise immunity for bumper readings */
    #define HALL1 analog(6)
    #define HALL2 analog(7)
    #define HALL3 IRDT[8]
    #define HALL4 IRDT[9]
    #define HALL5 IRDT[10]
    #define HALL6 IRDT[11]
    #define HALL7 IRDT[12]

    /************************* End of Constants ****************************/
    /*************************** Prototypes **********************************/
    void turn(void);
    void avoid(void);
    void mineDetect(void);
    void change_speed(void);
    void check_cds(void);
    void borderDetect(void);
    void init_CDS(void);
    void front_bump(void);
    void back_bump(void);
    /************************ End of Prototypes *****************************/
    /***************************** Globals **********************************/
    int irdr,irdl,irdc,rspeed,Rspeed= 0, lspeed,Lspeed = 0, test1 = 0 ;
    int hall1, hall2, hall3, hall4, hall5, hall6, hall7,
                                            hall8, hall9, hall10, hall11, hall12;
    int open=2600, close=2200, BBtime = 80;
     int CDSR_THRESHOLD,CDSL_THRESHOLD;
    /************************** End of Globals *******************************/

    void main(void)
    /****************************** Main **********************************/
    {
       init_analog();
       init_motortk();
       init_clocktk();
       init_serial();
       init_servos();

      /*Start MMH when front bumper is pressed*/
       while(front_bumper()<BUMPER_FUZZY_ZERO);
      IRE_OUT = IRE_ON;   /* Turn on the IR emitters */
       wait(1000);          /* Allow IR det & CDS cells to reach final values */

      init_CDS();     /* Find CDS threshold */

      lspeed = rspeed = FORWARD; /*Initial movement of TALRIK is forward*/
       change_speed();

       while(1)
       {
                  avoid();
                  mineDetect();
                  borderDetect();

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

    void avoid()
    {/*********************************************************************
     * Function:  Will avoid obstacles                              *
     * time                              *
     * Returns:  None                          *
     *                              *
     * Inputs                          *
     *   Parameters: None                          *
     *   Globals:    None                                  *
     *   Registers:  None                          *
     * Outputs                          *
     *   Parameters: None                          *
     *   Globals:    None                          *
     *   Registers:  None                          *
     * Functions called: turn                          *
     * Notes:                          *
     ********************************************************************/

                     /* The following block will read the IR detectors and decide
                       whether TALRIK needs to turn to avoid any obstacles */
                     irdr = IRR;
                     irdl = IRL-5;
```

```
                    irdc = IRC-13;

            if((irdr > IR_THRESHOLD)||(irdl > IR_THRESHOLD)||
              (irdc > IR_THRESHOLD))
        {
                IRE_OUT = LED_AVOID;
                if(irdc < irdl || irdc < irdr)
                {
                        if(irdr > irdl)
                        /* Start turning when something in front and keep turning
                                until nothing is in front */
                        {
                                lspeed = HREV;
                                rspeed = HFORW;
                        }
                        else
                        {
                                lspeed = HFORW;
                                rspeed = HREV;
                        }
                }
                else
                {       lspeed = HFORW;
                        rspeed = HREV;
                }

                change_speed();

                        while((irdr > IR_THRESHOLD ) || (irdl > IR_THRESHOLD )||
                             (irdc > IR_THRESHOLD))
                        {       irdr = IRR;
                                irdl = IRL-5;
                                irdc = IRC-13;

                                if(front_bumper()>BUMPER_FUZZY_ZERO) // Check bumper
                                {
                                        front_bump();
                                        break;
                                }
                                borderDetect();
                        }
                        wait(400);

                        IRE_OUT = IRE_ON;
        }
        lspeed = rspeed = FORWARD;
        change_speed();

                /* If the bumper is pressed, TALRIK will back up, and turn. */
                front_bump();
}


void turn()
/************************************************************************
 * Function:  Will turn in a random direction for a random amount of    *
 * time                                                  *
 * Returns:  None                                               *
 *                                                       *
 * Inputs                                                 *
 *   Parameters: None                                            *
 *   Globals:    None                                          *
 *   Registers:  TCNT                                              *
 * Outputs                                                *
 *   Parameters: None                                      *
 *   Globals:    None                                    *
 *   Registers:  None                                    *
 * Functions called: None                                    *
 * Notes:                                              *
 ***********************************************************************/
{
 int i;
 unsigned rand;

 rand = TCNT;

 if (rand & 0x0001)
 {
  lspeed = HREV;
  rspeed = HFORW;
  change_speed();
 }
 else
 {
  lspeed = HFORW;
  rspeed = HREV;
  change_speed();
 }

 i=((rand % 1024) + 35) * 2;
 wait(i);

}


void mineDetect()
{/***********************************************************************
 * Function:  Will detect magnets and drop a BB on it                          *
 * time                                                *
 * Returns:  None                                       *
 *                                                  *
 * Inputs                                              *
 *   Parameters: None                                      *
 *   Globals:    None                                                *
 *   Registers:  None                                    *
 * Outputs                                              *
 *   Parameters: None                                      *
 *   Globals:    None                                    *
 *   Registers:  None                                    *
 * Functions called: read_IR                                   *
 * Notes:                                              *
 ***********************************************************************/
                read_IR();
                hall1=HALL1;
                hall2=HALL2;
                hall3=HALL3;
                hall4=HALL4;
                hall5=HALL5;
                hall6=HALL6;
                hall7=HALL7;

                if(hall1 < 10 || hall2 < 10 || hall3 < 10 || hall4 < 10
                        || hall5 < 10 || hall6 < 10 || hall7 < 10)
                {
```

```
                                        IRE_OUT = LED_MINE_DET; // Turn on LED for mine detection
                                        lspeed = rspeed = STOP;
                                        change_speed();
                                        servo(SERVO1,open);
                                        wait(BBtime);
                                        servo(SERVO1,close);
                                        wait(2200);
                                        lspeed = rspeed = FORWARD;
                                change_speed();
                                wait(300);
                                IRE_OUT = IRE_ON;
                                }
}


void change_speed()
{/***********************************************************************
 * Function:  Will slowly change the speed of both motors.                *
 * time                                                       *
 * Returns:  None                                                *
 *                                                          *
 * Inputs                                                     *
 *   Parameters: None                                            *
 *   Globals:    None                                                           *
 *   Registers:  None                                          *
 * Outputs                                                     *
 *   Parameters: None                                            *
 *   Globals:    None                                          *
 *   Registers:  None                                          *
 * Functions called:                                                      *
 * Notes:                                                      *
 ***********************************************************************/
                while (lspeed != Lspeed || rspeed != Rspeed)
                {
                                if (Lspeed > lspeed) --Lspeed;
                                else if (Lspeed < ls peed) ++Lspeed;
                                if (Rspeed > rspeed) --Rspeed;
                                else if (Rspeed < rspeed) ++Rspeed;
                                motortk(LEFT_MOTOR,Lspeed);
                                motortk(RIGHT_MOTOR,Rspeed);
                                wait(1);
                }

}


void borderDetect()
{/***********************************************************************
 * Function:  Will check if robot is on border of search area and will  *
 *                                              turn in random direction if so.        *
 * Returns:  None                                                *
 *                                                          *
 * Inputs                                                     *
 *   Parameters: None                                            *
 *   Globals:    None                                                           *
 *   Registers:  None                                          *
 * Outputs                                                     *
 *   Parameters: None                                            *
 *   Globals:    None                                          *
 *   Registers:  None                                          *
 * Functions called: turn                                         *
 * Notes:                                                      *
 **************************************************************************/
                int i;
                IRE_OUT = LED_BORDER;
                read_CDS();    /* Read all 6 CDS photoresistor voltage dividers */
                if(CDSR < CDSR_THRESHOLD || CDSL < CDSL_THRESHOLD)
                {              motortk(LEFT_MOTOR, STOP);
                motortk(RIGHT_MOTOR, STOP);
                wait(100);
                lspeed = rspeed = HREV;
                change_speed();
         wait(1000);
                turn();
                wait(2000);
                lspeed = rspeed = FORWARD;
                change_speed();
                }
                IRE_OUT = IRE_ON;
}


void init_CDS()
{/**************************************************************************
 * Function:  Will configure the threshold value of the CDS cells      *
 *                                                                        *
 * Returns:  None                                    *
 *                                           *
 * Inputs                                       *
 *   Parameters: None                                *
 *   Globals:    None                                            *
 *   Registers:  None                               *
 * Outputs                                      *
 *   Parameters: None                                *
 *   Globals:    None                               *
 *   Registers:  None                               *
 * Functions called:                                       *
 * Notes:                                       *
 **************************************************************************/
                int i;
                for(i=0; i<30; i++)
                   { read_CDS();
                     wait(150);
                   }
                read_CDS();    /* Read all 6 CDS photoresistor voltage dividers */

                CDSR_THRESHOLD = CDSR * 0.70 ;
                CDSL_THRESHOLD = CDSL * 0.70 ;
}


void front_bump()
{/**************************************************************************
 * Function:  This checks the front bumper. If the bumper is pressed,  *
 *                                   MMH will back up, and turn.                    *
 *                                                                        *
 * Returns:  None                                    *
 *                                           *
 * Inputs                                       *
 *   Parameters: None                                *
 *   Globals:    None                                            *
 *   Registers:  None                               *
 * Outputs                                      *
 *   Parameters: None                                *
 *   Globals:    None                                *
 *   Registers:  None                                *
 * Functions called: change_speed                         *
 * Notes:                                        *
 ************************************************************************/
            if(front_bumper()>BUMPER_FUZZY_ZERO)
            {
                        motortk(LEFT_MOTOR, STOP);
                        motortk(RIGHT_MOTOR, STOP);
                        wait(300);
                        lspeed = rspeed = HREV;
                        change_speed();
                        wait(1000);
                        turn();
            }
}


void back_bump()
{/************************************************************************
 * Function:  This checks the back bumper. If the bumper is pressed,  *
 *                                  MMH will go forward, and turn.           *
 *                                                                          *
 * Returns:  None                                *
 *                                         *
 * Inputs                                    *
 *   Parameters: None                              *
 *   Globals:    None                                              *
 *   Registers:  None                              *
 * Outputs                                   *
 *   Parameters: None                              *
 *   Globals:    None                             *
 *   Registers:  None                              *
 * Functions called: change_speed                         *
 * Notes:                                        *
 ************************************************************************/
            if(rear_bumper()>BUMPER_FUZZY_ZERO)
            {
                        motortk(LEFT_MOTOR, STOP);
                        motortk(RIGHT_MOTOR, STOP);
                        wait(300);
                        lspeed = rspeed = HFORW;
                        change_speed();
                        wait(700);
                        turn();
            }
}
```

IR Test Code:

```
            /************************************************************************
             * Title        irtest2.c                       *
             * Programmer       Lee Cofer                                   *
             * Date         2/17/2002                          *
             * Version              2                                       *
             *                                                                   *
             * Description                                                          *
             *    This program tests the two IR detectors                    *
             ************************************************************************/
            /************************** Includes *******************************/
            #include <analog.h>
```

```
#include <clocktk.h>
#include <isrtk.h>
#include <serialtk.h>
#include <stdio.h>

#include <hc11.h>
#include <mil.h>
#include <vectors.h>
/************************ End of Includes ***************************/

/*************************** Constants ********************************/
#define IRR analog(2)
#define IRL analog(4)
#define IRC analog(5)
/************************ End of Constants ****************************/
void main(void)
/****************************** Main **********************************/
{
  int irdr,irdl,irdc;
  init_analog();
  init_clocktk();
  init_serial();
  *(unsigned char *) 0x7000=0x07; // Turn on the IR emitters
  while(1)
  {
        irdr = IRR;
        irdl = IRL-5;
        irdc = IRC-13;

        printf("R: {%d}",irdr);
        printf("  C: {%d}",irdc);
        printf("  L: {%d} \n",irdl);

        wait(300);
  }
}
/*************************** End of Main ******************************/
```

CdS Cell Test Code:

```
/***********************************************************************
 * Title      cdstest1.c                          *
 * Programmer      Lee Cofer                                        *
 * Date       4/12/2002                             *
 * Version         1                                                *
 *                                                                      *
 * Description                                                          *
 *   This program tests the two CdS Cells used on MMH.              *
 ***********************************************************************/
/*************************** Includes ********************************/
#include <tkbase.h>
#include <servotk.h>
#include <hc11.h>
#include <mil.h>
#include <vectors.h>
/************************ End of Includes ***************************/

/*************************** Constants ********************************/
#define CDSL CDS[0]
#define CDSR CDS[5]
#define IRE_OUT   *(unsigned char *)(0xffb9)
```

```
#define LED_BORDER 0x9f /* Border LED on */
#define IRE_ON 0x1f  /* IR emitters on */
#define IRE_OFF 0x00 /* IR emitters off */
/************************ End of Constants ****************************/
void main(void)
/****************************** Main **********************************/
{
  int cdsr,cdsl;
  init_analog();
  init_clocktk();
  init_serial();

  IRE_OUT = IRE_ON;

  while(1)
  {
          IRE_OUT = LED_BORDER;
   read_CDS();    /* Read all 6 CDS photoresistor voltage dividers */
          cdsr = CDSR;
          cdsl = CDSL;


          printf("{%d %d }\n",cdsr,cdsl);


          wait(150);
  }
}
/**************************** End of Main ******************************/
```

Hall Effect Sensor Test Code:

```
/***********************************************************************
 * Title       halltst3.c                        *
 * Programmer       Lee Cofer                                         *
 * Date        4/8/2002                           *
 * Version             3                                              *
 *                                                                        *
 * Description                                                            *
 *   This program tests the Hall Effect sensors used on MMH.          *
 ***********************************************************************/
/*************************** Includes ********************************/
#include <tkbase.h>
#include <hc11.h>
#include <mil.h>
#include <vectors.h>
/************************ End of Includes ***************************/
/*************************** Constants ********************************/
#define HALL1 analog(6)
#define HALL2 analog(7)
#define HALL3 IRDT[8]
#define HALL4 IRDT[9]
#define HALL5 IRDT[10]
#define HALL6 IRDT[11]
#define HALL7 IRDT[12]
/************************ End of Constants ***************************/
/*************************** Prototypes ********************************/
          void mineDetect(void);
/************************ End of Prototypes ****************************/
/***************************** Globals *********************************/
int hall1, hall2, hall3, hall4, hall5, hall6, hall7;
```

```
/************************** End of Globals ******************************/

void main(void)
/****************************** Main ***********************************/
{
  init_analog();
  init_clocktk();
  init_serial();

  while(1)
  {
        mineDetect();
  }
}
/**************************** End of Main ******************************/


void mineDetect()
{/********************************************************************
 * Function:  Will detect magnets and print out sensor results       *
 * time                                   *
 * Returns:  None                               *
 *                                      *
 * Inputs                                  *
 *   Parameters: None                           *
 *   Globals:    None                                     *
 *   Registers:  None                            *
 * Outputs                                 *
 *   Parameters: None                           *
 *   Globals:    None                          *
 *   Registers:  None                           *
 * Functions called: None                          *
 * Notes:                                   *
 ********************************************************************/

        read_IR();
        hall1=HALL1;
        hall2=HALL2;
        hall3=HALL3;
        hall4=HALL4;
        hall5=HALL5;
        hall6=HALL6;
        hall7=HALL7;

        printf("%d %d %d %d %d %d %d\n", hall1,hall2,
                        hall3, hall4, hall5, hall6, hall7);
        wait(40);
}
```

Servo Test Code:

```
/**********************************************************************
 * Title       servotst.c                      *
 * Programmer       Lee Cofer                           *
 * Date        3/16/2002                     *
 * Version           1                                  *
 *                                                     *
 * Description                                         *
 *   This program tests the servo that controls the door of the BB     *
 *         Dropping Mechanism of my special sensor suite.          *
 *                                    *
 ***************************************************************************/
/*************************** Includes ********************************/
#include <analog.h>
#include <clocktk.h>
#include <isrtk.h>
#include <serialtk.h>
#include <stdio.h>
#include <servotk.h>
#include <hc11.h>
#include <mil.h>
#include <vectors.h>
/************************ End of Includes ****************************/
/*************************** Constants ********************************/
#define SERVO1 0
#define HALL1 analog(4)
/************************ End of Constants ****************************/


void main(void)
/****************************** Main **********************************/
{
  int open=2600, close=2200, hall1,time;
   init_analog();
   init_clocktk();
   init_serial();
   init_servos();
   while(1)
   {        printf("Enter time: ");
                    time = read_int();
                    servo(SERVO1,open);
                    wait(time);
                    servo(SERVO1,close);
   }
}
/*************************** End of Main *****************************/
```