MURRY!

Jon Preussner
4/17/02
EEL5666
Final Report

Table of Contents

Abstract

Murry is a coin-collecting robot. He searches for coins, picks them up, sorts them, and displays the coin type on an LED array. Collision avoidance is provided by 2 IR emitter/detectors and 4 bump sensors on the front and rear bumpers. While wandering around, Murry uses a metal detector circuit search for coins on the floor.

Executive Summary

Murry was designed to find and identify coins on smooth, hard-surfaced floors. He was designed to run indoors. He has 3 primary parts: the metal detector, the coin-fetching arm, and the coin sorter. Upon detecting metal with the metal detecting circuit, Murry backs up and picks the coin up with his coin-fetching arm. The coin is then dropped into the sorter, sorted, and identified. The coin type is then displayed on an LED array. If no coin was retrieved, an error message is displayed.

The brains of the robot are provided by an MRC11 with the MRSX01 sensor expansion. The robot uses 2 hacked servomotors for motion. Actuation for the coin-fetching arm is provided by 2 Futaba S3003 servos. Collision avoidance is provided by 4 bump sensors and 2 Sharp IR emitter/detector pairs. The robot is powered by 3 separate battery sources. The main battery pack consists of 8 AA Ni-Cad rechargeable batteries. This battery pack provides power to the microcontroller, servos, motors and sensors. There is additional 9V battery used to power the metal detector circuit. The third battery pack is 2 C batteries. It is used to power the sorter light. Murry uses 5 CdS sensors to determine the type of coin that has been dropped into the sorter.

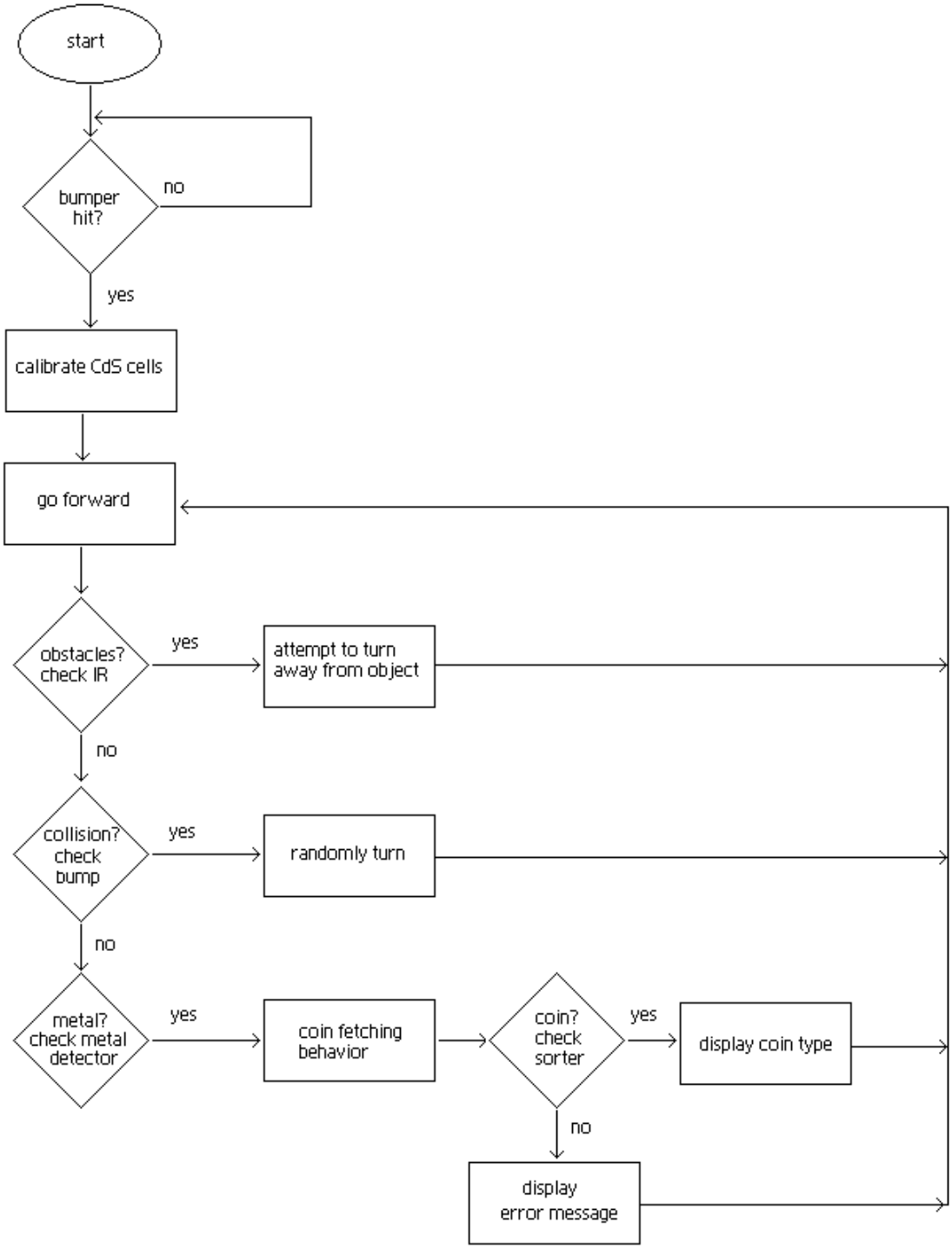
Introduction

The task of picking up coins off hard-surfaced floors is a tedious and sometimes painful process for humans. It requires bending over, which can cause back pain after just several coins. Picking up coins is also frustrating if you don't have long nails to get under the coin. Automating this process using an autonomous agent saves time, injury and frustration. Murry picks up coins off of hard surfaced floors.

This paper describes how Murry functions. It describes his platform, actuation, sensor suite, behaviors, experimental layout and results. It also includes an appendix with source code and parts listing. While constructing Murry I used several sources for information. I used the Talrik assembly manual and the MRSX01 assembly manual written by Keith Doty. Both are available from the Mekatronix website (http://www.mekatronix.com).

Integrated System

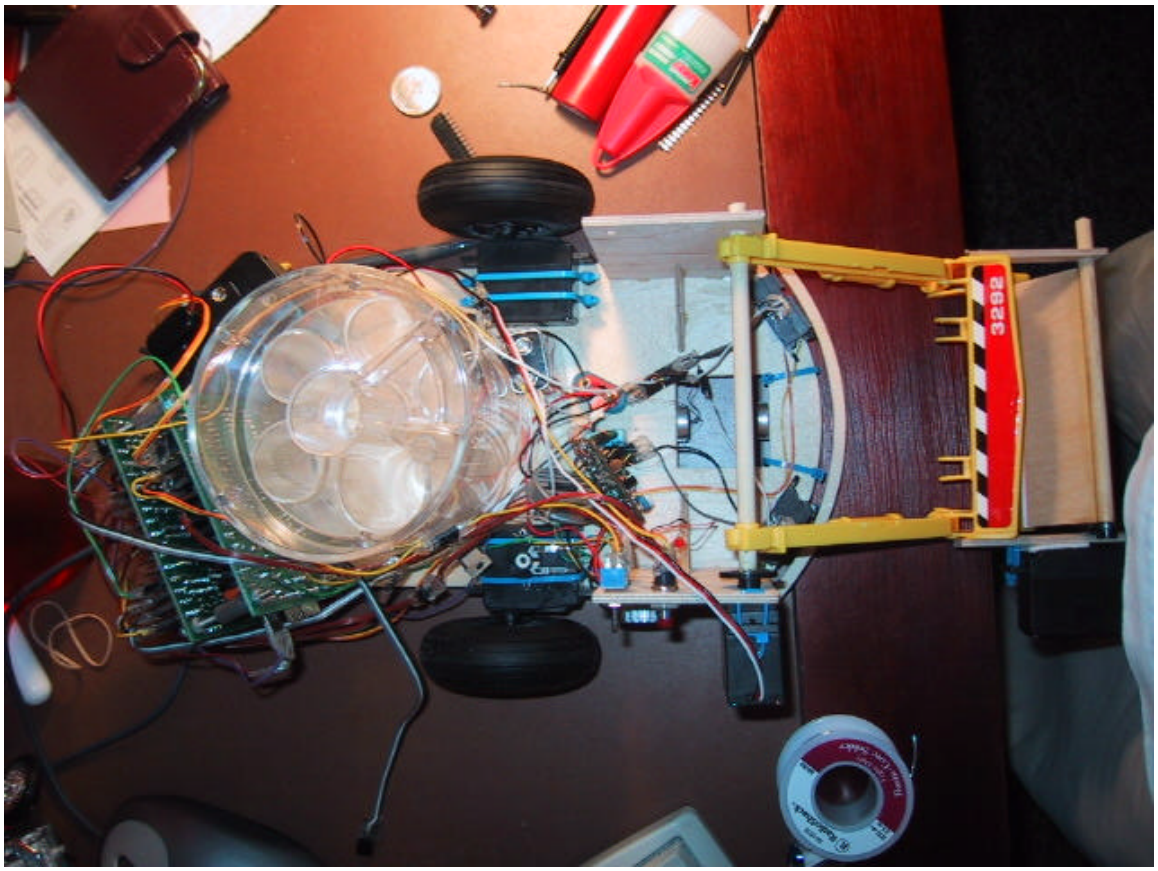
Murry is an autonomous coin-collecting robot. Here is a flowchart of his behaviors.



This flowchart shows how Murry functions. He waits until the bumper is hit. Then calibrates the CdS cells on the sorter. Then he begins his wander behavior. He avoids

obstacles and searches for coins. Once a coin is found he goes into his fetch and sort behaviors. After the coin is indicated on the LED array he goes beck into wander mode. These behaviors meet the specifications for Murry's functioning.

Mobile Platform



Top view of the platform layout

I designed Murry's platform to be simple. It is the shape of a long rectangle with rounded ends. Bump sensors are attached to the ends for collision avoidance. Two IR emitter/detector pairs are attached in the front of the platform. They are angled outwards for collision avoidance.

Murry's arm is supported by 2 pillars near the front of the platform. The arm is connected to a dowel rod on a servo. The arm is raised and lowered by the turning of this servo. The trapdoor on the arm is connected to a dowel rod on a servo. It is opened and closed by the turning of this servo. The 2 wheels are located near the center of the robot to accommodate easy turning. There is a skid near the rear of the platform, which Murry rests his weight on.

I took the special sensor into account when designing my platform layout. Any kind of metal or strong EM field by the detector's coil could throw off the sensor. In order to avoid this problem I mounted the coil as close to the front as possible, and placed the MRC11, MRSX01, and battery pack as far back as possible.

Actuation

Murry uses 2 servos and 2 hacked servomotors. Two Futaba S3003 servos are used to move the coin-fetching arm. They are both stock. One controls the raising/lowering of the arm. The other controls the opening/closing of the trapdoor. Both servos are hooked up to the MRSX01's servo outputs. Software was written to move the servos slower than normal (see slowServo() in appendix). This was done to avoid damage to the arm. Two hacked servomotors are used to move the robot. They were originally Futaba S148 servos. A full servo hack (PC board was removed) was performed on the servos. They are driven by the motor pins of the MRSX01.

Sensors

Murry's sensor suite consists of IR sensors, bump sensors, CdS cells, and a metal detector circuit. Sharp IR emitter/detector pairs are angles out from the front of the robot. They provide collision avoidance. They are powered by a 5V DC voltage regulator and are connected to the IR sensor pins of the MRSX01. Four bump sensors are positioned on the bumpers of the robot for collision avoidance. They are connected to the bump sensor resistor network on the MRSX01. Five CdS cells are positioned around the coin sorter for coin identification. The CdS cells are connected to the MRSX01's CdS cell pins. They are self-calibrated before the robot begins wandering. This software can be found in cal_cds() in the appendix.

Special Sensor

metal detector out-of-the-box

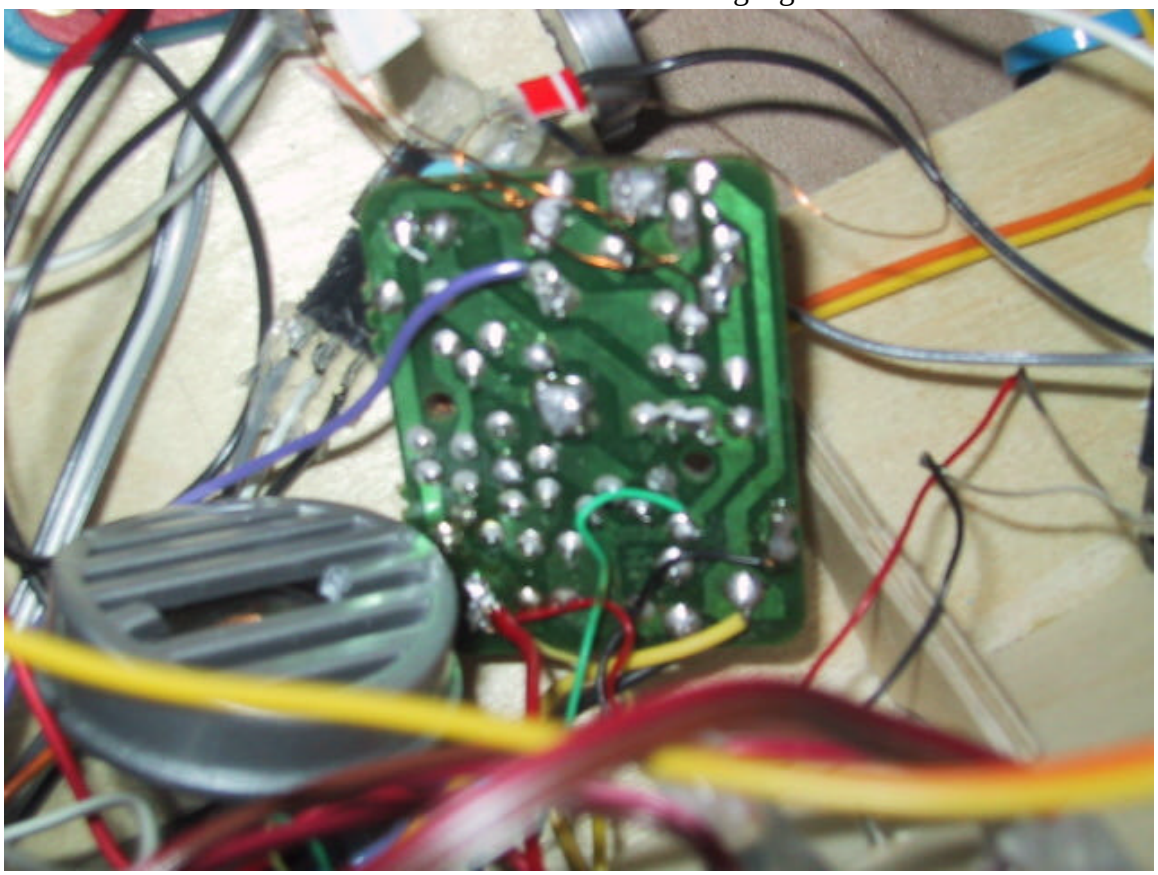
The circuit is a metal detector toy from the Discovery Channel store in Schaumburg, Illinois. The toy costs $10 and has since been discontinued. Luckily I was able to purchase a second detector before they left the shelves in Schaumburg. I was worried about the IR sensors throwing off the metal detector readings, but during testing I noticed no effect with both systems running at the same time. I initially tested the detector as it was out of the box. It was able to detect all the coins within a close proximity (1-2cm away) without being thrown off by large metal objects far away. After disassembling the detector I began to search for a signal to send to the microcontroller.

I initially was going to use the pulses sent to the detector's speaker to trigger the coin-fetching behavior. After analyzing the signal on the oscilloscope I found the pulses to be about 4ms apart when metal was under the coil. Writing software for this signal would be cumbersome. I would have to use the HC11's input capture system to count the clock cycles between speaker pulses. If the duration between pulses were short enough the coin-fetching behavior would be called.

Upon further analysis of the metal detector's circuitry I found an analog signal, which ranged from 0.6V to 2.5V (30 to 128 for sensor reading) when metal was under the coil. Finding such a signal was a great relief. I immediately tested the signal on the

oscilloscope and found it to be a smooth dc voltage. I decided to use this signal to elicit the coin-fetching behavior.

The metal detector was originally a handheld toy powered by a 9V battery. After discarding the non-functional parts, I mounted the coil and sensitivity potentiometer. I decided to power the sensor with a 9V battery rather than running it off of the robot's main battery pack. In order to do this I needed to make a common ground between the 2 sources. I soldered a wire between the negative terminals of both batteries to do this. I then soldered a wire to the location I had found the analog signal on the circuit earlier.



Close up view of the metal detector, and analog signal wire (purple wire)

Behaviors

Murry has 4 behaviors: calibrate, wander, fetch, and sort. When Murry is turned on he waits for a bumper to be hit. Once he detects a bumper has been hit he calibrates the CdS cells on the coin sorter. Murry then begins to wander, looking for coins. He will avoid objects using IR sensors. If he hits an object he will turn randomly and continue searching for a coin. When Murry detects a coin he will attempt to collect it using a scoop on an arm. Once the coin has been collected it is dropped into a sorter. The sorter is surrounded by 5 CdS cells that make it possible to identify the type of coin retrieved. The type of coin is displayed on an LED array.

Experimental Layout and Results

While developing Murry I tested his characteristics and recorded them. The following figure is a screenshot from the main test program I wrote for Murry. The software was writing in the ICC11 IDE and downloaded to the robot using the 68HC11 High Speed Downloader.

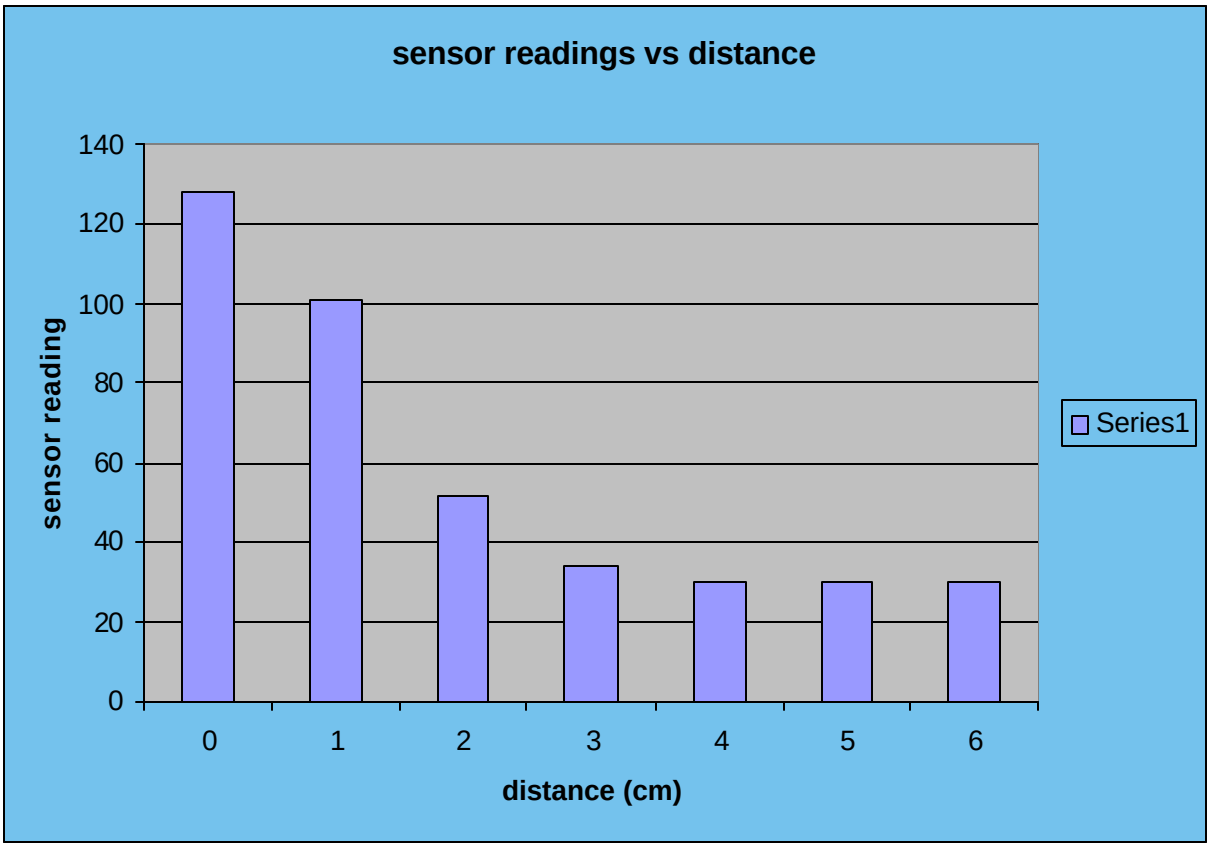
Murry's self-calibrating CdS software (found in appendix) worked well in any ambient lighting environment. Here is an example of its versatility.

```
Thresholds set:
CDS[1]= 22
CDS[2]= 29
CDS[3]= 141
CDS[4]= 92
CDS[5]= 76
Thresholds set:
CDS[1]= 29
CDS[2]= 33
CDS[3]= 143
CDS[4]= 93
CDS[5]= 80
```

CDS calibration with little ambient light

CDS calibration with a lot of ambient light

I also collected data on the robot's metal detector. Here is a table describing distance vs sensor value.



The data was nearly identical for all coins tested.

Conclusion

In conclusion Murry was a great success. He effectively collects, sorts and identifies coins. However I did have to cut out some functionality due to time constraints. Originally an LCD panel was to be used to display a total of change collected. The LCD panel was built but the code I didn't have enough time to write code for it. I wisely substituted a LED array for it, losing the ability to display a total. There are also areas where the design could be improved. If I were to start this project over I would have made the platform larger. Everything fit on the 7" by 11" platform that I'm using now, but it is cluttered a larger platform would have made it easier to work on the robot. The metal detector is finicky (being that it is a kid's toy) and is difficult to calibrate for coins. A more accurate metal detecting circuit would be a definite improvement. Also the coin fetching arm sometimes misses coins. I felt this wasn't a big deal since Murry will simply attempt to pick up the coin again when he runs over it. The most helpful caveat I learn during the robot building process was K.I.S.S. – Keep It Simple Stupid. By breaking the process down to simple steps I was able to efficiently build this robot. If I had more time to spend on this robot there are several things I would add. I would like to use a relay instead of a bump switch to turn on the sorter light. The bump sensor works but a relay would be more reliable. Adding self-calibration to the metal detector would also be nice. This could be done by using an a/d port to find the proper voltage level for the sensitivity knob instead of using the potentiometer.

Documentation

Special Thanks to: TaeHoon Choi, Aamir Qaiyumi, Dr. Arroyo, Dr Schwartz

References:

Keith L. Doty.MRC11 Assembly Manual; Mekatronix 1999.
Keith L. Doty.MRSX01 Assembly Manual; Mekatronix 1999.
Keith L. Doty.Talrik Assembly Manual; Mekatronix 1999.
Oopic.com Sharp emitter/detector pinouts(http://www.oopic.com/gp2d12.htm)

Sources for Parts

MRC11 and MRSX01
Source: Mekatronix (www.mekatronix.com)
Price: $85 and $60

Coin sorter
Source: Wal-Mart
Price: $5

Coin-fetching arm (from bulldozer toy)
Source: Toy's-R-Us
Price: $15

Futaba S148 servos
Source: Tower Hobbies (www.towerhobbies.com)
Price: $30 each

Futaba S3003 servos
Source: Tower Hobbies (www.towerhobbies.com)
Price: $24 each

Sharp IR emitter/detector pairs
Source: Mekatronix (www.mekatronix.com)
Price: $10 each

Metal Detector
Source: Discovery Channel Store
Price: $10

Appendix

Program code in C
Main Program

```
/*************************************************************************
Main program for murry
 *************************************************************************/

/*************************** Includes *********************************/
#include <tkbase.h>
/************************ End of includes *****************************/

/*************************** Constants ********************************/
#define FORWARD 100
#define REVERSE -100
#define HFORW 50
#define HREV -50
#define IR_THRESHOLD 117
#define METAL_ON 70
#define BUMPER_FUZZY_ZERO 12 /* Noise immunity for bumper readings */
#define HEAD_SERVO 0 /*This is Servo_01 connector on MRSX011 board*/
#define LEDS   *(unsigned char *)(0xffb9)


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

/*************************** Prototypes *******************************/
void turn(void);
void move_head(int);
void slowServo(int servo, int from, int to);
void fetch(void);
void sorter(void );
void cal_cds(void );
/************************ End of Prototypes ***************************/

/*************************** Globals **********************************/
int temp;
unsigned int cds1=0, cds2=0, cds3=0, cds4=0, cds5=0;

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


void main(void)
/*************************** Main *************************************/
{
unsigned int IR_delta[NIRDT], IR_Threshold[NIRDT];
int i, fb, rspeed, lspeed, delta_rspeed, delta_lspeed;

  init_analog();
```

```
  init_motortk();
  init_clocktk();
  init_servos();
  init_serial();

  lspeed = rspeed = HFORW; /*Initial movement of TALRIK is forward*/

  LEDS=(0x00);
  wait(300);                    /* Allow IR detectors to reach final values */

 /*Start TALRIK moving forward when back bumper is pressed*/
  while(rear_bumper()<BUMPER_FUZZY_ZERO);

  cal_cds();
  motortk(RIGHT_MOTOR, rspeed);
  motortk(LEFT_MOTOR, lspeed);

  while(1)
  {

 /* The following block will read the IR detectors and decide
    whether TALRIK needs to turn to avoid any obstacles
 */

     read_IR();


  if((IRDT[0] > IR_THRESHOLD ) || (IRDT[1] > IR_THRESHOLD ))
    {
      if(IRDT[0] > IRDT[1])
      /* Start turning when something in front and keep turning until nothing
        is in front */
        {
            lspeed = HREV;
            rspeed = HFORW;
           }
      else
           {
             lspeed = HFORW;
             rspeed = HREV;
           }

      motortk(RIGHT_MOTOR, rspeed);
      motortk(LEFT_MOTOR, lspeed);

      while((IRDT[IR10clk] > IR_THRESHOLD ) || (IRDT[IR11clk] > IR_THRESHOLD ) ||
           (IRDT[IR12clk] > IR_THRESHOLD ) ||  (IRDT[IR1clk] > IR_THRESHOLD ) ||
           (IRDT[IR2clk] > IR_THRESHOLD ))
                    read_IR();

    }
```

```
  else
    {
      lspeed = 100;
      rspeed = 100;
    }

    motortk(RIGHT_MOTOR, rspeed);
    motortk(LEFT_MOTOR, lspeed);

// This "if" statement checks the front bumper. If the bumper is pressed,
//   TALRIK will back up, and turn.


    if((rear_bumper()>77)&&(rear_bumper()<154))
    {
    printf("backup\n");
      motortk(LEFT_MOTOR, HREV);
      motortk(RIGHT_MOTOR, HREV);
      wait(350);
      turn();
    }

    if((rear_bumper()>19)&&(rear_bumper()<64))
    {
    printf("go forward\n");
      motortk(LEFT_MOTOR, HFORW);
      motortk(RIGHT_MOTOR, HFORW);
      wait(350);
      turn();
    }

    if(IRDT[2]>METAL_ON)
    {
    printf("metal\n");

    fetch();

          }
    wait(35);
  }
}
/*************************** End of Main ******************************/

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)
 {
   motortk(RIGHT_MOTOR, FORWARD);
   motortk(LEFT_MOTOR, REVERSE);
 }
 else
 {
   motortk(RIGHT_MOTOR, REVERSE);
   motortk(LEFT_MOTOR, FORWARD);
 }
 /*for (i = 0; i < rand; i++);*/
 i=(rand % 1024) + 35;
 wait(i);

}
/*************************end turn************************************/


void move_head(int fb)
/*****************************************************************************
 * Function:  Will turn TALRIK head to face direction of bump contact.  *
 * time                                                       *
 * Returns:  None                                                *
 *                                                          *
 * Inputs                                                     *
 *   Parameters: fb front bumper value.                              *
 *   Globals:    None                                          *
 *   Registers:  None                                           *
 * Outputs                                                    *
 *   Parameters: None                                            *
 *   Globals:    None                                          *
 *   Registers:  None                                           *
 * Functions called: None                                        *
 * Notes:                                                     *
 ****************************************************************************/
{
```

```
   switch (fb)/*Select an IR detector on TALRIK according to clock position */
     {
       case 127: servo(HEAD_SERVO,1400);  break;
       case 151: servo(HEAD_SERVO,1725);  break;
       case 81:  servo(HEAD_SERVO,2050);  break;
       case 103: servo(HEAD_SERVO,2375);  break;
       case 45:  servo(HEAD_SERVO,2700);  break;
       case 61:  servo(HEAD_SERVO,3025);  break;
       case 23:  servo(HEAD_SERVO,3350);  break;
       case 36:  servo(HEAD_SERVO,3675);  break;
       case 16:  servo(HEAD_SERVO,4000);  break;
     }

}
/*************************end move_head************************************/

/*************************slow servo***************************************/
void slowServo(int a, int b, int c)
{
        if(b>=1000 || b<=5000)
        {
                servo(a,b);
                wait(500);
        }
        temp=b;
        while(c!=temp && c>=1000 && c<=5000)
        {
                if(temp<c)
                {
                temp+=20;
                servo(a,temp);

                }
                else
                {
                temp-=20;
                servo(a,temp);

                }

                wait(50);
//              if(c%50==0)
//              printf("servo at: %d\n", temp);
        }
//      while(1);
}
/*************************end slowServo************************************/

/*************************fetch********************************************/

void fetch(void)
```

```
{
        motortk(RIGHT_MOTOR, 0); //stop first
    motortk(LEFT_MOTOR, 0);
    wait(500);

        motortk(RIGHT_MOTOR, HREV); //backup then stop
    motortk(LEFT_MOTOR, HREV);    //so coin is in front of scoop
    wait(1000);
    motortk(RIGHT_MOTOR, 0);
    motortk(LEFT_MOTOR, 0);

    slowServo(1, 1300, 4260); //lower scoop


        motortk(RIGHT_MOTOR, HFORW); //move forward a bit then stop
    motortk(LEFT_MOTOR, HFORW);
    wait(200);

    motortk(RIGHT_MOTOR, HFORW); //move forward a bit then stop
    motortk(LEFT_MOTOR, HFORW);
    wait(200);

    motortk(RIGHT_MOTOR, 0);
    motortk(LEFT_MOTOR, 0);

    slowServo(0, 2500, 4500); //close scoop

        slowServo(1, 4260, 2700); //raise scoop
        slowServo(0, 4500, 2500); //open scoop
        slowServo(1, 2700, 1300); //raise scoop
        servo(1,1050);

        sorter();          //CALL SORTER
        slowServo(1, 1000, 1300);  //this turns off sorter light


}
/*************************end fetch*************************************/

/*************************sorter*************************************/

void sorter(void )
{
unsigned int temp_one;
unsigned int temp_two;
unsigned int temp_three;
unsigned int temp_four;
unsigned int temp_five;

int i, holdon;
int counter=0, coin=0;
```

```
wait(440); //let cds cells stableize

//while(holdon>CDS_ONE)
//{
//read_CDS();
//holdon=CDS[1];
//}


while(counter<4000)
{

        read_CDS();


 temp_one=CDS[1];
 if((temp_one < cds1 ))
 {
        coin=1;
        printf("CDS[1]: %u counter: %d\n", temp_one, counter);
        counter=0;
 }

 temp_two=CDS[2];
 if((temp_two < cds2 ))
 {
        coin=2;
        printf("CDS[2]: %u counter: %d\n", temp_two, counter);
        counter=0;
 }

 temp_three=CDS[3];
 if((temp_three < cds3 ))
 {
        coin=3;
        printf("CDS[3]: %u counter: %d\n", temp_three, counter);
        counter=0;
 }

 temp_four=CDS[4];
 if((temp_four < cds4 ))
 {
        coin=4;
        printf("CDS[4]: %u counter: %d\n", temp_four, counter);
        counter=0;
 }

 temp_five=CDS[5];
 if((temp_five < cds5 ))
```

```
 {
        coin=5;
        printf("CDS[5]: %u counter: %d\n", temp_five, counter);
        counter=0;
 }

 counter++;
}

printf("coin %d\n", coin);
switch (coin) {
                case 0:
                        LEDS=(0x15);
                        wait(300);
                        LEDS=(0x0a);
                        wait(300);
                        LEDS=(0x15);
                        wait(300);
                        LEDS=(0x0a);
                        wait(300);
                        LEDS=(0x15);
                        wait(300);
                        LEDS=(0x0a);
                        wait(300);
                        LEDS=(0x00);
                        break;
                case 1 :
                        LEDS=(0x01);
                        wait(1500);
                        LEDS=(0x00);
                        break;
                case 2 :
                        LEDS=(0x02);
                        wait(1500);
                        LEDS=(0x00);
                        break;
                case 3 :
                        LEDS=(0x04);
                        wait(1500);
                        LEDS=(0x00);
                        break;
                case 4 :
                        LEDS=(0x08);
                        wait(1500);
                        LEDS=(0x00);
                        break;
                case 5 :
                        LEDS=(0x10);
                        wait(1500);
                        LEDS=(0x00);
                        break;
```

```
                default:
                        LEDS=(0x15);
                        wait(300);
                        LEDS=(0x0a);
                        wait(300);
                        LEDS=(0x15);
                        wait(300);
                        LEDS=(0x0a);
                        wait(300);
                        LEDS=(0x15);
                        wait(300);
                        LEDS=(0x0a);
                        wait(300);
                        LEDS=(0x00);
                        break;
                }

}
/**************************end sorter*************************************/

/**********************cal_cds ***************************************/
void cal_cds(void )
{
        int i,temp1=0, temp2=0, temp3=0, temp4=0, temp5=0;
        servo(1,1050);  //this turns on sorter light
        wait(440);                  //wait for light to turn on

        for(i=0 ; i<30; i++)
        {
        read_CDS();
        temp1+=CDS[1];
        temp2+=CDS[2];
        temp3+=CDS[3];
        temp4+=CDS[4];
        temp5+=CDS[5];
        wait(40);
        }
        cds1=(temp1/30-5);
        cds2=(temp2/30-5);
        cds3=(temp3/30-5);
        cds4=(temp4/30-5);
        cds5=(temp5/30-5);

        printf("Thresholds set: \nCDS[1]= %u\nCDS[2]= %u\nCDS[3]= %u\nCDS[4]=
%u\nCDS[5]= %u\n", cds1, cds2, cds3, cds4, cds5);

        slowServo(1, 1000, 1300);  //this turns off sorter light
}
/**************************endcal_cds************************************/
```