Date:	09 Dec 1996
Student Name:	Kevin McFarlin
TA:	Scott Jantz
Instructor:	Keith L. Doty

University of Florida
Department of Electrical Engineering

EEL 5934
Intelligent Machines Laboratory

Big Mac: The Migrating Robot

Table of Contents

Abstract

Big Mac is an autonomous migrating robot. It has four types of sensors -- tactile sensor, IR emitter/detectors, tilt sensor, and digital compass. The robot uses these sensors to integrate four different behaviors -- bump reaction, collision avoidance, tilt reaction, and migration.

Executive Summary

The purpose of this project was to design a migrating robot which could negotiate obstacles during the migration. The robots name is Big Mac. Big Mac's microprocessor is a Motorola 68HC11 with a Novasoft ME11 expansion board. The program code was written and compiled in ICC11 format.

Big Mac's mobile platform is octangular, and the robot rides on a pair of toy tank treads which are actuated by two servos. Big Mac uses four different types of sensors to interact with the environment. These sensors are tactil bump sensors, IR emitter/detectors, tilt sensor, and a digital compass. These sensors allow Big Mac to implement four different behaviors. These behaviors are bump reaction, collision avoidance, tilt reaction, and migration. The first three behaviors mentioned serve to allow Big Mac to *survive* in an environment. The migration behavior is the algorithm that gives Big Mac its purpose.

All four of Big Mac's behaviors operate properly. However, it is the integration of these four behaviors that allow Big Mac to perform adequately in a complex environment.

Introduction

The purpose of this project was to produce a robot which is intelligent enough to roam an area avoiding any obstacles which it may encounter. Furthermore, the robot must be capable of maintaining its orientation in any environment. The robot could then reference its orientation in order to migrate to one end of the room. This paper discusses how this was achieved.

The first topic of discussion is the integrated system. Next, the method of robot actuation will be discussed and the topic of the robot's necessary sensors will follow. The paper will close with the behavior algorithms implemented in Big Mac and its performance in a complex environment.

Integrated System

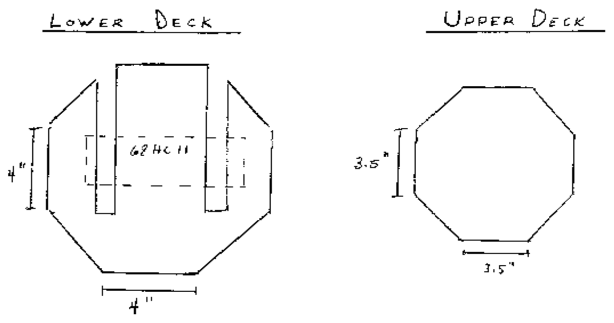
Big Mac's microprocessor is a Motorola 68HC11 with evaluation board. The microprocessor is used in expanded mode in conjunction with the Novasoft ME11 expansion board. The program code was written and compiled in ICC11 format. Big Mac's power supply consists of eight AA rechargeable batteries.

Big Mac possesses four behaviors -- bump reaction, collision avoidance, tilt reaction, and migration. Big Mac must integrate all four of its behaviors in order to achieve its goal of reaching the north end of an enclosed area. The bump reaction and collision avoidance are the fundamental necessities of an intelligent robot. Big Mac must be capable of negotiating any obstacles that it encounters in order to perform the migration behavior properly.

Mobile Platform

The robot sits on a pair of treads salvaged from a toy tank. On top of the tread system is an octangular platform. This lower deck platform serves as the basis for mounting the 68HC11 EVBU board and other essential circuitry. The platform has two 1 inch slots cut for tread clearance and several holes drilled for routing wires around the platform. Above the lower deck is another octangular platform called the upper deck. This deck serves as a cover for the circuitry and provides a mounting area for an elevated bump sensor. The upper deck is also a level platform for mounting the digital compass. The deck isolates the compass from any metal or circuitry which may influence its accuracy. A detailed diagram of the mobile platform is given in Figure 1 below.

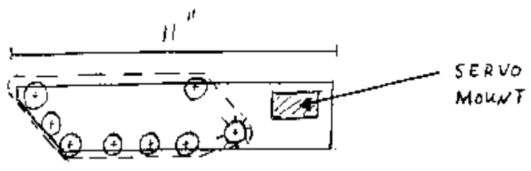
Figure 1



Actuation

The robot rests on a pair of treads with a wheel base of 4.5 inches. The treads have a row of fingers which fit into grooves in the tread guides. These fingers serve to keep the treads aligned and on the tread guides. The treads are driven by servos. A Large gear -- 50 teeth -- is epoxied to each servo. Then a smaller gear -- 20 teeth -- is epoxied to each sprocket that rotates the tank treads. These gears are meshed together to actuate the treads. This gearing system is necessary to speed up the movement of the treads because the sprocket that rotates the treads is only 0.75 inches in diameter. Figure 2 is a detailed diagram of the tread system.

Figure 2



SIDE VIEW OF TREAD SYSTEM

Sensors

Tactile Sensor

Big Mac has a two level bump sensor. One level is on the lower deck of the mobile platform, and the second level is on the upper deck of the mobile platform. The sensors are made from single-pole single-throw momentary contact keyboard switches and a large, stiff nylon zip-tie. The switches are glued around the front periphery of the platforms, and the zip-tie is glued to the tip of the switches. The Figure 3 contains a diagram of the bump sensor design. Figure 4 is the circuit used to interface the bump sensor with the 68HC11.

Figure 3

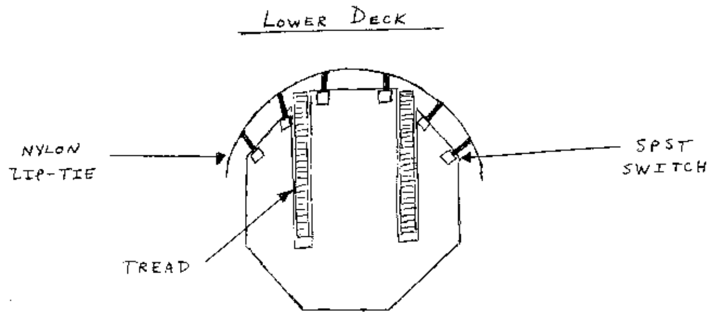
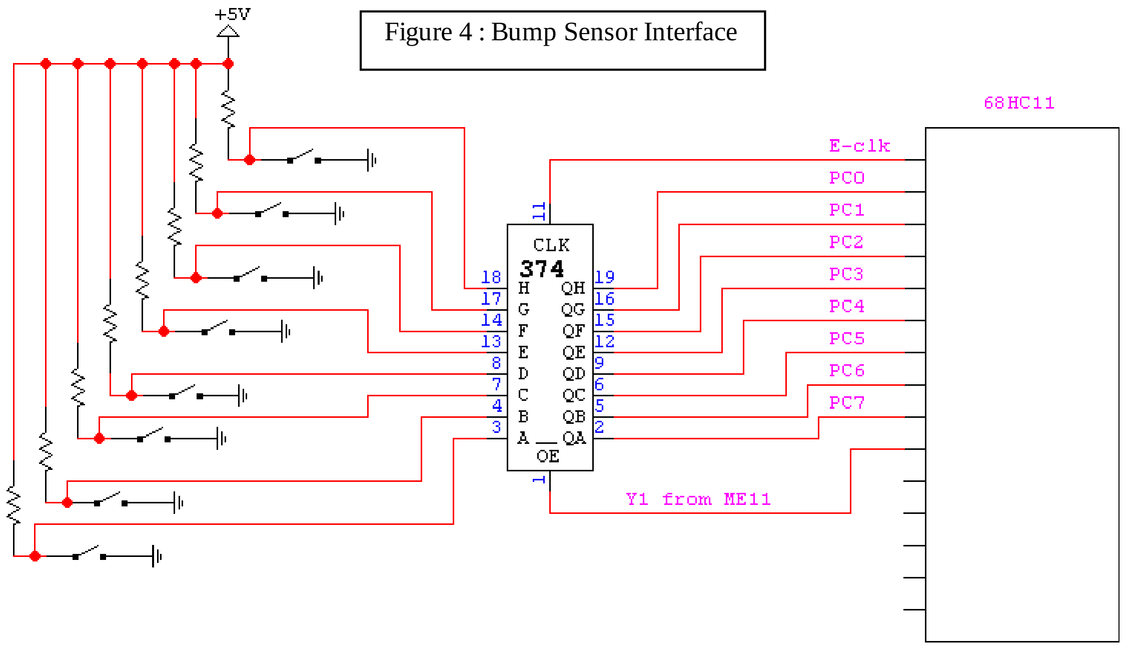


Figure 4 : Bump Sensor Interface



Big Mac has the capability to sense which side the bump occurs on. Both the upper level and the lower level bump sensors prove to be sensitive. They are an effective secondary collision avoidance sensor.

Infrared Emitter/Detector Pair

The infrared sensors consist of two components -- the IR emitter and the IR detector. The IR emitters are IR LED's which are glued to the underside of the mobile platform. The underside of the platform is painted black to cut down on the reflection of IR light from Big Mac's body. This reduces the amount of false triggering in the IR detector. The IR light is modulated at 40 kHz, and the Sharp IR detectors only detect 40 kHz modulated light. This serves to filter out any light from sources other than Big Mac's IR emitters. These sensors perform as effective primary collision avoidance sensors.

Tilt Sensor

The tilt sensor is a mercury switch from an old pin ball machine. The tilt sensor is implemented exactly like the bump sensors. Only six bump sensors are used in the diagram given in Figure 1. Therefore, the tilt switch was added on to the same circuitry. The tilt sensor bit is bit seven of address (4000)hex of the microprocessor. Bit zero through bit five of address (4000)hex is reserved for the bump sensor.

Digital Compass

The digital compass is a design by Kevin McFarlin and Jeff Webb. It was made from a PC mouse, rare earth magnets, and a 1 inch fishing bobber floating in a 35 mm film canister. A detailed description of the design is included in the appendix. The compass is the sensor that provides the robot with its relative orientation in an area. The information given to the robot by the compass allows Big Mac to implement its migration behavior.

Behaviors

Bump Reaction

When Big Mac senses a bump with its tactile sensor, the robot can sense which direction the bump came from and then decide how to react. If the bump is on the far left side, Big Mac will back up and turn slightly to the right to negotiate the obstacle. If the bump is sensed on the near left, Big Mac will back up and turn hard to the right. The robot will react similarly to bumps on the right side as well.

Big Mac also has a short term memory when it bumps into an object. If the robot experiences a bump several times at approximately the same compass coordinate, it will react in a different way from the previous times the same bump was experienced. For example, if Big Mac enters a corner in which the IR collision avoidance fails, it will generally bump on the left and then bump on the right and then bump on the left again. This sort of oscillation would continue if Big Mac did not have a short term memory. However, the robot can remember the compass coordinates of its previous two bumps and take this into consideration when reacting to the current bump.

Collision Avoidance

The IR emitter / detector pairs are the sensors used to implement this behavior. First, the speed of Big Mac is a function of the maximum IR reading. The relationship is as follows:

$$\textit{base speed} = \frac{[(100 - IRmax) * 4]}{100} + 30$$

where IR max is the maximum IR reading normalized from 0 to 100. When the robot does not *see* anything -- IR max is zero -- then the speed is 70. 70 is the duty cycle of the motors. When Big Mac is within approximately 6 inches of an object IR max approaches 100. This yields a speed of 30. The relationship between IR readings and speed allow Big Mac to slow down when approaching an object. This also prevents the robot from bumping into an object at full speed.

Big Mac has four IR detectors that it utilizes for collision avoidance. There are two outer detectors -- the outer left detector and the outer right detector. There are also two inner detectors -- the inner left detector and the inner right detector. Big Mac's turn rate is dependent on the difference between these emitters. Big Mac uses two differences -- the difference between the two inner detectors and the difference between the two outer detectors. The relationship is as follows:

$$\textit{turn rate} = \frac{Kouter * (outerRightIR - outerLeftIR)}{100} + \frac{Kinner * (innerRightIR - innerLeftIR)}{100}$$

where Kouter is the outer IR turn rate gain and Kinner is the inner IR turn rate gain. The turn rate represents the difference in the right and left motor speed from the robots base speed. For example, a turn rate of 35 results in the left motor being at a duty cycle of 35 above the base speed and the right motor being at a duty cycle of 35 below the base speed. This yields a right turn. Furthermore, a turn rate of -35 results in the left motor being at a duty cycle of 35 below the base speed and the right motor being at a duty cycle of 35 above the base speed. This yields a left turn. Therefore, one can understand that there are many varying turn rates and because of the feedback from the IR differences, Big Mac can have a smooth turning reaction when encountering obstacles.

Tilt Reaction

Because Big Mac has a tread system with relatively high clearance, the robot has a tendency to drive up on low lying objects without detecting them with either the bump sensors or the IR detectors. This can prove to be a problem because Big Mac could either stall its motors or flip over and turtle itself. Another problem is that the digital compass does not give an accurate reading unless the robot is level. Therefore, the tilt sensor can be used to avoid these undesired situations. When Big Mac experiences a tilt greater than 30 degrees, it will disable the other behaviors, back up, and then continue in a random direction.

Migration

The digital compass allows Big Mac to migrate to the relative north end of a room or area. Relative north is defined as the direction that Big Mac is pointing when powered up or reset. Big Mac attempts to migrate in this direction whenever an obstacle is not within the immediate path of the robot. If an object is encountered, the robot will negotiate the obstacle and then -- after the obstacle is avoided -- the robot will begin to migrate again. This behavior causes the robot to drift in a general direction. Inevitably, Big Mac will reach the furthest point north in a room. The turn rate of the robot is dependent on the compass reading only when an obstacle is not eminent. The relationship is as follows:

$$\text{migrate rate} = \text{turn rate} = \frac{\text{Kcompass} * (\text{Heading Difference})}{100}$$

where the heading difference is the difference between the relative north heading and Big Mac’s current compass heading and Kcompass is the compass feedback gain. Refer to the section on collision avoidance for a detailed description on the turn rate significance.

Performance

Bump Reaction

The performance of the bump reaction algorithm is functional. The short term bump memory improves the performance by preventing the robot from becoming trapped in narrow corridors. Without the short term memory or some method of randomness, the robot would oscillate right and left because it would continue to bump the left wall and then the right wall. However, the implementation of the digital compass provides Big Mac with a reference point for which to remember where previous bumps occurred.

A drawback to the bump sensor is that it protrudes from the sides of the robot. This proves to be a problem when the robot is in a cluttered environment where items can snag the bump sensor. In the future, a more streamlined sensor needs to be designed.

Collision Avoidance

The infrared collision avoidance algorithm operates proficiently. It has the capability of making gradual corrections in the robots course and speed to maneuver around objects. However, the algorithm also works efficiently when Big Mac is approaching objects head on. The algorithm first slows down the robot and then makes the corrections necessary to negotiate the obstruction quickly.

The behavior does have problems with black objects. Yet, the algorithm is successful with most dark items other than black. There is also problems with transparent or shiny, metallic items. However, these items are nearly always detected by the bump sensors.

Tilt Reaction

The tilt algorithm serves its purpose to protect the robot from motor stall and turtling. The algorithm is simple, yet useful for a tertiary obstacle avoidance method.

Migration

The migration algorithm is the most unique behavior implemented in Big Mac. The robot clearly tries to reach the north side of the room. There are times when the migration behavior conflicts with the collision avoidance behavior. Yet, Big Mac is persistent and inevitably finds a way to migrate. This behavior can be a useful technique in getting a robot to a certain destination. The robot does not actually have a clear destination, however, if there is a north wall with a recessed area in the wall Big Mac will eventually find its way to the recess. This is mainly due to the migration behavior and randomness generated by the orientation of objects in the room. In the future, a recessed wall such as the one mentioned above can be the location of a charging station or item drop off point. So, this migration behavior can be implemented in many useful ways.

Conclusion

Big Mac overall was a success. The hardware proved to be a rather sturdy design. At first, there were doubts about the gear and tread system. However, the treads have been operating since the beginning of the project and there has been no instance of failure. The software of Big Mac proved to be adequate, yet there are still some algorithms and behaviors which need to be tested and implemented.

Some lessons learned from this project are that careful planning is required to realize a significant goal, and that goal needs to be specified from the start. This will give the designer a more structured purpose to pursue. Furthermore, a designer needs to pay close attention to detail. This sort of design approach will save time in the long run. In closing, Big Mac has been an educational project and has laid a firm foundation for more in-depth machine intelligence work in the future.

References

Jeff Webb is the author of the following ICC11 library files used by Big Mac:

multitsk.c	-- multitasking routine
timekeep.c	-- time delay routine
motcontr.c	-- servo control routine
compass.c	-- compass data capture routine

Appendix A: Big Mac Program Code

```
/***********************************************************************
***/
/* Robot:                  BigMac                              */
/* Algorithms Implemented:    Differential IR steering, bump avoidance, */
/*                         migration to relative north           */
/***********************************************************************
***/

/*----------------------------------------------------------------------*/
/* Include files                                           */
/*----------------------------------------------------------------------*/
 #include <multitsk.h>
 #include <timekeep.h>
 #include <serio.h>
 #include <motcontr.h>
 #include "sensors.c"
 #include <function.h>
 #include <compass.h>

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

```
/* Defines                                              */
/*----------------------------------------------------------------------*/
  #define motorTimeConstant 5
  #define motorSamplingPeriod 50
  #define left 0
  #define right 1

/* Bump Flag                                              */
int urgentBump = 0;

/* Bump Variables                                              */
unsigned char bump;
int bumpSpeed;
int bumpRate;

/* IRVariables                                              */
int irMax;
int irSpeed;
int irRate;
int dOir;
int dIir;
int Kinner;
int Kouter;
```

```
/* Migration Variables                                        */

int head;

int dHead;

int dHeading;

int Kcomp = 60;

int migrateRate;




/*********************************************************************

***/

/*                                                    */

/* Bump Maneuvering: this routine takes over when a bump occurs          */

/*                                                    */

/*********************************************************************

***/


void moveBump(int direction, int lotime, int hitime)

{

  bumpSpeed = 0;

  bumpRate = 0;

  urgentBump = 1;

  setDesiredSpeed(bumpSpeed);
```

```
setDesiredTurnRate(bumpRate);
delay(150);

if (direction == left)
{
  bumpSpeed = -75;
  bumpRate = -35;
  setDesiredSpeed(bumpSpeed);
  setDesiredTurnRate(bumpRate);
  delay(random(lotime,hitime));
};
if (direction == right)
{
  bumpSpeed = -65;
  bumpRate = 35;
  setDesiredSpeed(bumpSpeed);
  setDesiredTurnRate(bumpRate);
  delay(random(lotime,hitime));
};

bumpSpeed = 0;
bumpRate = 0;
setDesiredSpeed(bumpSpeed);
```

```
  setDesiredTurnRate(bumpRate);
  delay(150);

  bumpSpeed = 70;
  bumpRate = 0;
  setDesiredSpeed(bumpSpeed);
  setDesiredTurnRate(bumpRate);

  urgentBump = 0;
}

/*************************************************************************
***/
/*                                                    */
/* Bump Avoidance: arbitrates how to react to the bump                    */
/*                                                    */
/*************************************************************************
***/
void avoidBump()
{
  while(1)
  {
    bump = getBumpValue();
```

```
   if (bump == 0) urgentBump = 0;
   if (bump != 0)
   {
     urgentBump = 1;
     if (bump == 1) moveBump(right,500,750);
     else if ((bump != 1) && (bump <= 3)) moveBump(right,1000,1500);
     else if ((bump > 3) && (bump <= 7)) moveBump(right,1500,2500);
     else if ((bump > 7) && (bump <= 8)) moveBump(left,2000,2500);
     else if ((bump > 8) && (bump <= 31)) moveBump(left,1500,2000);
     else if ((bump > 31) && (bump <= 63)) moveBump(left,750,1000);
    };
  };
}

/*********************************************************************
***/
/*                                                      */
/* IR Avoidance: uses the difference between the two outer ir and the     */
/*            and the difference between the two inner ir to steer the */
/*            robot.  The speed is a function of the maximum ir reading*/
/*********************************************************************
***/
void diffIR()
```

```
 {
  while(1)
  {
    int i;
    int ir[6];
    for(i=0; i<6;i++)
      ir[i] = getIR(i);

/*Find maximum of ir array*/
    irMax = max(ir,6);

/*Determine ir differential*/
    dOir = ir[outerLeft] - ir[outerRight];
    dIir = ir[innerLeft] - ir[innerRight];

/*Determine ir turn rate gains*/
    if ((irMax > 50) && (abs(dOir) < 10))
      Kouter = 700;
    else
      Kouter = 50;
    if ((irMax > 50) && (abs(dIir) < 10))
      Kinner = 1000;
    else
```

```
       Kinner = 75;


/*Compute ir speed*/

     irSpeed = ((100 - irMax) * 7) / 10 + 20;


/*Compute ir turn rate*/

     irRate = Kouter * dOir / 100 + Kinner * dIir / 100;

    };

}


/***********************************************************************

***/

/* Compass migration routine: the robot uses the compass to migrate to    */

/*                          relative north.  Relative north is defined  */

/*                          by the direction the robot is pointed when  */

/*                          reset is pressed.                          */

/***********************************************************************

***/


void migrate()

{

  while(1)

  {
    head = getHeading();

    dHead = 360 - head;

    if (dHead > 180)

      dHeading = dHead - 360;

    else

      dHeading = dHead;


/*Compute migration turn rate*/

     migrateRate = Kcomp*dHeading/100;

  };

}


/**********************************************************************
***/

/*    Debugging: prints to screen the crucial parameters                  */

/**********************************************************************
***/


void debug()

{

  while(1)

  {

     printf("Ready...\n");
    printf("IR Speed:       %d \n",irSpeed);
    printf("IR Rate:        %d \n",irRate);
    printf("Migration Rate: %d \n",migrateRate);
    printf("Heading:        %d \n",head);
    printf("dHeading:       %d \n",dHeading);
    printf("Bump:           %d \n",bump);
    printf("Urgent Bump:    %d \n",urgentBump);
    home();
  };
}


/*******************************************************************
***/
/* Main Program                                                   */
/*******************************************************************
***/


main()
  {


/*  Initializations                                              */
     initTimeKeeper();
     initMultiTasking();
```

```
/*  Initialize I/O devices                                  */
      initSensors();
      initCompass();

/*  Initialize Control Processes                            */
      initMotionControl(motorSamplingPeriod, motorTimeConstant);

/*  Start Multitasking processes                            */
      startProcess(*motionControl);
      startProcess(*sampleSensors);
      startProcess(*avoidBump);
      startProcess(*diffIR);
      startProcess(*migrate);
      startProcess(*debug);

/*  Main Program Loop                                       */
  while(1)
  {

/*   If ir rate is small and there is no bump the robot can migrate      */
     if ((urgentBump == 0)&&(abs(irRate) < 15 ))
     {
```

```
    setDesiredTurnRate(migrateRate);

    setDesiredSpeed(irSpeed);

    };


/*   If ir rate is high and there is no bump the robot must use ir rate   */

    if ((urgentBump == 0)&&(abs(irRate) >= 15))

    {

    setDesiredTurnRate(irRate);

    setDesiredSpeed(irSpeed);

    };

  };

}




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

/* Set up interrupt vectors                                              */

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

#include <vectors.c>


/*********************************************************************
***/

/* Sensor Interface                                         */

/*                                                          */
/* Programmer:   Kevin McFarlin                                  */
/* Date:         November 20, 1996                               */
/* Version:      1                                               */
/*********************************************************************
***/


/*---------------------------------------------------------------------*/
/* Includes                                                            */
/*---------------------------------------------------------------------*/
  #include <ir.h>
  #include <bump.h>


/*---------------------------------------------------------------------*/
/* Defines                                                             */
/*---------------------------------------------------------------------*/


/* Delay between sensor readings in milliseconds                       */
     #define sensorSampleRate 100


/* IR Sensor Names                                                     */
     #define outerLeft 0
     #define innerLeft 1
     #define innerRight 2
     #define outerRight 3
     #define leftWall 4
     #define rightWall 5

/* Bump Sensor Names                                    */
     #define farLeft 0
     #define nearLeft 1
     #define leftCenter 2
     #define rightCenter 3
     #define nearRight 4
     #define farRight 5

/* Turn Names                                           */
     #define right 1
     #define left 0

/*------------------------------------------------------------------------*/
/* Init Sensors Process                                 */
/*------------------------------------------------------------------------*/
void initSensors()
  {
    initIR(6, 90, 128, 100, 0xFF);
    initBump(6);
  }


/*---------------------------------------------------------------------*/
/* Sample Sensors Process                                         */
/*---------------------------------------------------------------------*/
void sampleSensors()
  {
    while(1)
      {
          sampleIR();
          sampleBump();
          delay(sensorSampleRate);
      };
  }




/***********************************************************************
***/
/*                                          */
/* Motion Control System                          */
/*                                          */
```

```
/**********************************************************************
***/

/*--------------------------------------------------------------------*/
/* Include files                                                  */
/*--------------------------------------------------------------------*/
  #include "motor.c"
  #include <timekeep.h>

/*--------------------------------------------------------------------*/
/* Motion Control Defines                                         */
/*--------------------------------------------------------------------*/
  #define left 0
  #define right 1

/*--------------------------------------------------------------------*/
/* Motion Control Private Variables                               */
/*--------------------------------------------------------------------*/
  int desiredSpeed = 0;
  int desiredTurnRate = 0;
  int leftSpeed = 0;
  int rightSpeed = 0;
  int motorDT = 50;
  int motorDV = 10;


/*----------------------------------------------------------------------*/
/* Set motor speed sampling period                              */
/*----------------------------------------------------------------------*/
void setMotorDT(int newDT)
  {
    motorDT = newDT;
  }


/*----------------------------------------------------------------------*/
/* Set increment for changing motor speed                          */
/*----------------------------------------------------------------------*/
void setMotorDV(int newDV)
  {
    motorDV = newDV;
  }


/*----------------------------------------------------------------------*/
/* Set desired speed function                                   */
/*----------------------------------------------------------------------*/
void setDesiredSpeed(int newSpeed)
  {
    desiredSpeed = newSpeed;
  }


/*----------------------------------------------------------------------*/
/* Set desired turn rate function  (right turn +)                        */
/*----------------------------------------------------------------------*/
void setDesiredTurnRate(int newRate)
  {
    desiredTurnRate = newRate;
  }


/*----------------------------------------------------------------------*/
/* Get current speed function                                           */
/*----------------------------------------------------------------------*/
int getDesiredSpeed()
  {
    return desiredSpeed;
  }


/*----------------------------------------------------------------------*/
/* Get current turn rate function                                       */
/*----------------------------------------------------------------------*/
int getDesiredTurnRate()
```

```
  {
    return desiredTurnRate;
  }


/*---------------------------------------------------------------------*/
/* Initialize Motion Control System                              */
/*---------------------------------------------------------------------*/
void initMotionControl()
  {
/*  Initialize Variables                                    */
      desiredSpeed=0;
      desiredTurnRate=0;
      leftSpeed = 0;
      rightSpeed = 0;
      motorDT = 50;
      motorDV = 10;


/*  Initialize the time keeper                               */
      initTimeKeeper();


/*  Initialize the motors                                    */
      init_motors();
```

```
/*  Turn off motors                                    */

      motor(left,0);

      motor(right,0);


   }


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

/* Motion Control Process                                 */

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


void motionControl()

  {

/*  Variables                                        */

      int leftDutyCycle;

      int rightDutyCycle;

      int desiredLeftSpeed;

      int desiredRightSpeed;


/*  Main motion control loop                          */

      while (1)

        {


/*        motion control system                      */
```

```
         desiredRightSpeed = desiredSpeed - desiredTurnRate;
         desiredLeftSpeed  = desiredSpeed + desiredTurnRate;

/*       motion smoothing routines - slow response                    */
         if (desiredLeftSpeed - leftSpeed > 0)
           leftSpeed += motorDV;
         else if (desiredLeftSpeed - leftSpeed < 0)
           leftSpeed -= motorDV;

         if (desiredRightSpeed - rightSpeed > 0)
           rightSpeed += motorDV;
         else if (desiredRightSpeed - rightSpeed < 0)
           rightSpeed -= motorDV;

/*       motor speed control system  (does nothing yet)               */
         leftDutyCycle  = leftSpeed;
         rightDutyCycle = rightSpeed;

/*       motor driver                                     */
         motor(left,  leftDutyCycle );
         motor(right, rightDutyCycle);

/*       Wait                                         */
```

```
      delay(motorDT);


   };

}
```

Appendix B : Digital Compass Design

Introduction

The objective of this sensor design is to provide an accurate method for monitoring robot rotation. There are existing sensors for this purpose such as rate gyros and digital compasses. However, these sensors are expensive -- in the range of $100. The sensor design addressed in this report -- a digital compass -- is about one-tenth of the cost of a typical market sensor. Furthermore, because the digital compass will be implemented on a robot, the design needs to be compact. This can accomplished by mounting the sensor in a 35 mm film canister with the interface circuitry on a two inch by one inch printed circuit board.

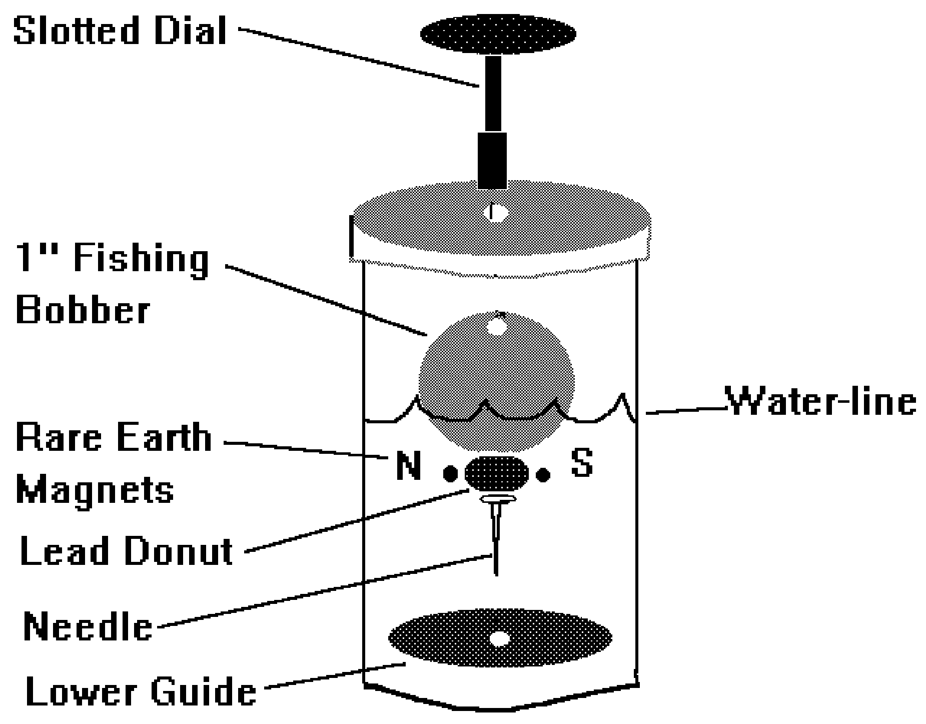
The digital compass design was realized by hacking an inexpensive Logitech PC mouse. The hack allows the use of a precision slotted wheel encoder to monitor the orientation of a rare earth magnet which is suspended in a 35 mm film canister. Another useful feature of the PC mouse hack is the serial interface with the microprocessor. Only one data line is required to transmit data to the 68HC11.

Sensor Description

Mechanical Design

The physical design of the digital compass is light-weight and compact. The magnet and bobber assembly is contained in a 35 mm film canister. A diagram of the sensor is shown in Figure 1. Two rare earth magnets are epoxied to a lead donut, and this lead donut attaches to the bottom of the fishing bobber. The slotted encoder wheel then attaches to the top of the bobber and projects through the lid of the film canister. A lower and upper guide are necessary to center the bobber and the encoder wheel. These guides also serve to restrict any movement perpendicular to the axis of the assembly. The lower guide is fashioned from hard plastic and has a hole in its center to align the needle projecting from the underside of the bobber. The upper guide -- not shown in Figure 1 -- is a guide from the inside of the mouse housing. This guide fits the shaft of the encoder wheel. A second compass was designed using a needle shaft and a slightly different upper guide.

Figure 4: Compass Mechanical Design



On one corner of the PC mouse printed circuit board, an IR emitter/detector pair extends from the board. The IR emitter/detector pair was cut away from the printed circuit board. Then the pair was positioned on the lid of the film canister and in-line with the slots on the encoder wheel. Hot glue held the IR emitter detector pair in place and wires were soldered onto the pair to connect it with the original PC mouse printed circuit board.

Microprocessor Interface

The compass uses the electronics from a Microsoft two button mouse to communicate with the Motorola 68HC11 microprocessor via the processor's Serial Communications Interface (SCI) system. The serial interface makes this sensor very simple to interface, and very universal. The compass was designed for use with a M68HC11 microprocessor, but it can be interfaced to almost any computer system that has an asynchronous serial port.

Serial Protocol

A standard two button mouse uses the serial transmission protocol shown in Figure 2. Each mouse transmission consists of a series of three data bytes that tell the amount and direction of rotation for each of the mouse's two encoder wheels. The mouse only transmits data when movement or a button action has been detected.

Figure 5: Mouse Transmission Protocol

```
Microsoft Mouse Operation


Serial UART: 1200 baud, data=7, stop=1, parity=none


Mouse Protocol of Transmission
        bit:    7  6  5  4  3  2  1  0
byte 1  (sync)  0  1  L  R y7 y6 x7 x6
byte 2  (dX)    0  0 x5 x4 x3 x2 x1 x0
byte 3  (dY)    0  0 y5 y4 y3 y2 y1 y0


Notes:  - all dx, dy, are two's complement binary numbers
```

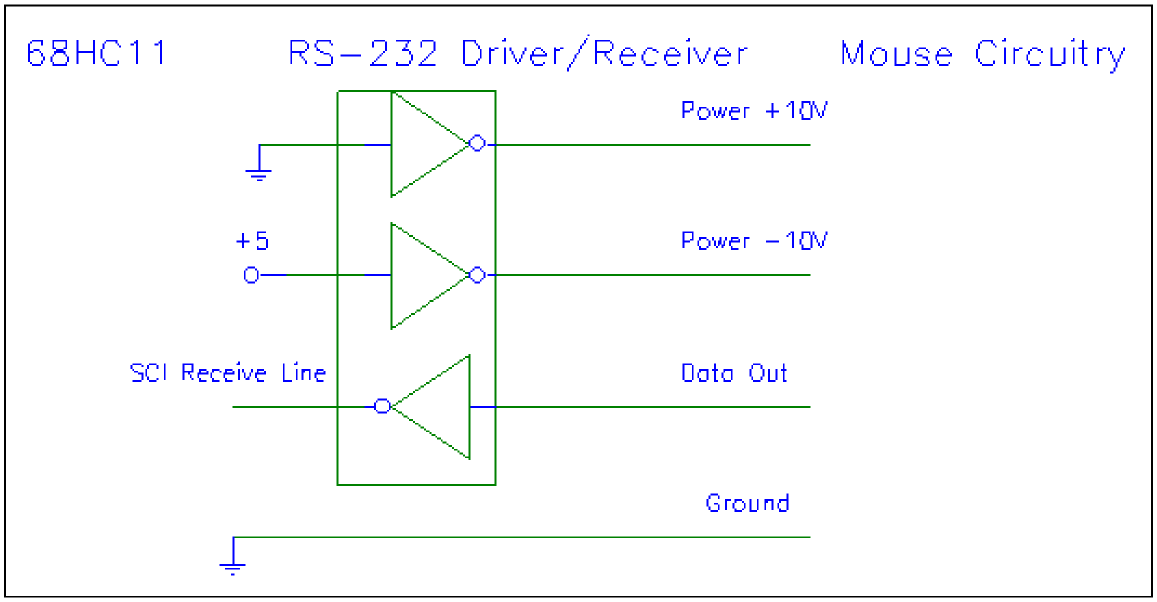
The 68HC11 does not directly support the mouse's "7N1" serial protocol, so the SCI interface uses an "8N1" protocol instead. Fortunately, this protocol mismatch works just fine, and bit 8 of each data byte is always received as a logic "1".

Hardware Interface

The mouse transmits data using RS-232 voltage levels, so we used the MC145407 RS-232 driver/receiver on the Motorola EVBU board for voltage level conversion. The EVBU board uses one driver for serial transmission, and one receiver for serial reception. We placed a mechanical switch in series with the RS-232 receive line so that the user may select the source of the signal entering the 68HC11 SCI receive pin. The switch selects either the compass or the EVBU DB25 connector, which is used for PC communication.

The microprocessor also needs to supply power to the mouse hardware. The mouse uses two RS-232 lines (one +10 V, and one -10 V) as power supplies. We used the two unused line drivers on the MC145407 chip to provide these voltage levels. A block diagram of the hardware interface is shown in Figure 3.

Figure 6: Hardware Interface for Compass Electronics



Software Interface

The software interface for the compass consists of an initialization routine for enabling the compass, an interrupt service routine for handling the incoming data, and an output variable that contains the current heading. The initialization routine configures the SCI system to interrupt the processor on each incoming data byte. This means that no polling is required, and the processor will only be interrupted when the compass heading changes. The interrupt service routine (ISR) keeps track of which data byte is being processed and reads the data byte from the SCI port. The ISR then does some bit manipulation to piece together the change in heading information, and then calculates the new compass orientation.

Experimental Procedure

We tested the compass by rotating it to several headings and measuring the error in the compass readings. This was done by marking headings on a test stand, rotating the compass to each heading, and recording the compass reading. This allowed us to determine the average errors for each heading orientation.

We then tested the compass for cumulative error by rotating the compass many revolutions in one direction. We also checked the compass for tilt error, although we did not do a formal experiment.

Results

The data from the compass tests are shown in Tables 1 and 2. Figures 4 and 5 show plots of the average error in the compass reading for each test heading. The compass seemed to have a large error at some orientations. The cause of this error could be a slight compass tilt, or some magnetic attraction, but we are not certain. The smaller errors in most of the readings are due to friction or encoder resolution (about 2 degrees per pulse of the encoder).

Table 1: Compass #1 Test Data

Actual Heading	Trial 1	Trial 2	Trial 3	Trial 4	Avg. Error
0	358	0	358	0	1
30	24	24	21	26	6.25
60	59	51	53	53	6
90	90	80	80	78	8
120	120	101	103	101	13.75
150	147	130	143	130	12.5
180	178	176	172	176	4.5
210	218	206	206	206	3
240	247	239	237	239	3
270	273	270	268	266	2.5
300	302	298	302	300	1.5
330	331	331	333	331	1.5
360	358	0	0	358	1

Figure 7: Compass #1 Error Plot

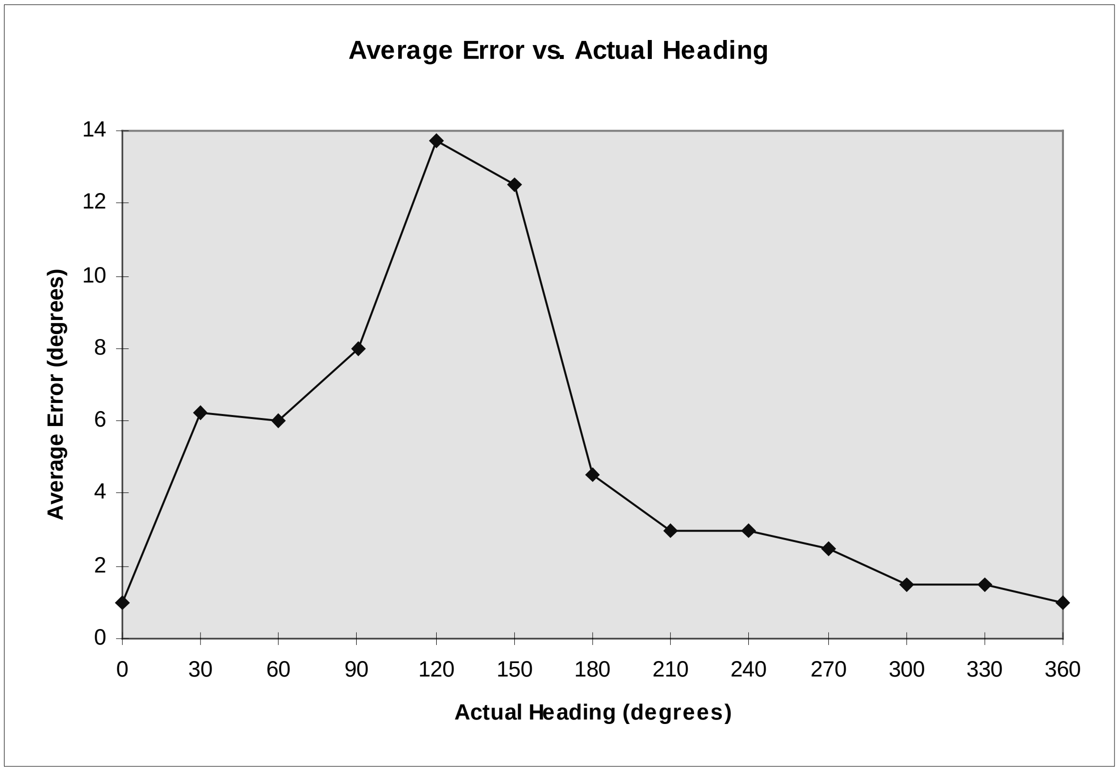
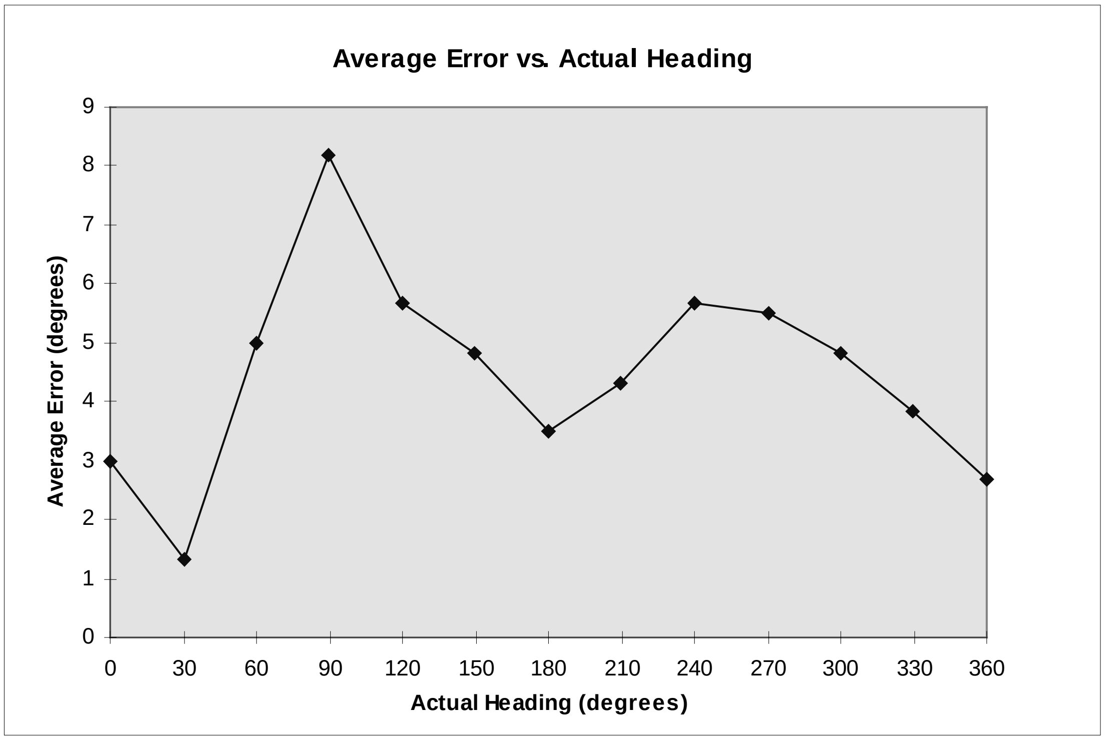


Table 2: Compass #2 Test Data

Actual Heading	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Avg. Error
0	0	0	358	358	358	348	3
30	28	32	26	30	30	28	1.33
60	67	63	65	67	70	65	5
90	99	105	95	105	95	99	8.17
120	128	132	124	130	126	122	5.67
150	151	160	151	162	155	149	4.83

180	178	187	183	189	180	178	3.5
210	206	220	210	220	212	206	4.33
240	235	250	233	247	233	243	5.67
270	262	277	266	281	262	273	5.5
300	296	308	296	304	291	304	4.83
330	321	333	321	333	325	327	3.83
360	0	0	358	358	354	354	2.67

Figure 8: Compass #2 Error Plot



The compass showed no cumulative error over many rotations, so it worked well in this respect. The compass has a very large error when it is tilted from vertical, so this is a great consideration in mounting and using the compass. The compass also uses very strong magnets, so it must be mounted away from any type of ferrous materials.

Conclusion

The digital compass design yielded mixed results. The design was successful in several areas. The design was inexpensive – approximately $12 – and very compact. The compass was also easy to interface with the microprocessor and required minimal processing time. However, the compass had some problems with accuracy. This was largely due to friction on the compass guides and imprecise assembly techniques. The performance could be improved with a more refined upper guide design and the use of precision tools to construct the sensor. Although we did not meet our design goals in terms of accuracy, the digital compass should still be useful in many applications.

Appendix : ICC11 Interface Routine

```
/**********************************************************************/
/* Compass Interface Routines                                         */
/**********************************************************************/

/*--------------------------------------------------------------------*/
/* Include files                                                      */
/*--------------------------------------------------------------------*/
#include "serial.c"

/*--------------------------------------------------------------------*/
/* Global Output Variables                                                   */
/*--------------------------------------------------------------------*/

/* Current Heading                                                    */
     unsigned int heading=0;

/* Current Speed                                                      */
     int speed = 0;

/*--------------------------------------------------------------------*/
/* Private Variables                                                  */
/*--------------------------------------------------------------------*/

/* Current data byte                                                  */
     int byteNumber=1;

/* Current Pulse Count                                                */
     int pulseCount=0;

/* Pulses per compass revolution                                      */
     pulsesPerRev = 172;

/*--------------------------------------------------------------------*/
/* Initialization Routine                                             */
```

```
/*----------------------------------------------------------------------*/
void initCompass()
  {
/*  Initialize variables                                                */
      byteNumber = 1;
      pulseCount=0;
      heading = 0;

/*  Set up serial port                                                  */
      BAUD = Baud1200;
      SCCR1 = Prot8N1;
      SCCR2= PollT_IntR;
  }

/*----------------------------------------------------------------------*/
/* Interrupt Service Routine                                            */
/*----------------------------------------------------------------------*/

#define SCI_ISR compassISR
#pragma interrupt_handler compassISR

void compassISR(void)
  {
/*  Check if we have an incoming data byte                              */
      if (SCSR & RDRF)
        {

/*        Change in heading                                             */
            static char dx=0;

/*        Read incoming data byte                                       */
            char inByte = SCDR;

/*        Process current data byte                                     */
            switch (byteNumber)
              {
                case 1 :
                  dx = (inByte & 0x03) << 6;
                  break;
                case 2 :
                  dx = dx + (inByte & 0x7F);
                  pulseCount = pulseCount + dx;
                  if (pulseCount < 0)
                    pulseCount = pulsesPerRev + pulseCount;
                  else if (pulseCount > pulsesPerRev-1)
                    pulseCount = pulseCount - pulsesPerRev;
                  speed = -dx;
                  break;
                case 3 :
                  break;
                default:
                  break;
              };

/*        Update byte counter                                             */
            byteNumber++;
            if (byteNumber>3 || byteNumber < 1)
              byteNumber = 1;
        }
      else
        {
/*        Clear interrupt flag                                            */
            int inByte = SCDR;
        };
  }

/**************************************************************************/
/* Compass Test Program                                                   */
/**************************************************************************/


/*------------------------------------------------------------------------*/
/* Include files                                                          */
/*------------------------------------------------------------------------*/
#include <stdio.h>

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

```
/* Test Routine                                                        */
/*---------------------------------------------------------------------*/
main()
  {
    int oldPulses;

/*  Set up compass and enable interrupts                               */
      initCompass();
      printf("The Compass is ready...\n");
      INTR_ON();

/*  Continuously Display Compass Data                                  */
      oldPulses=pulseCount;
      while(1)
        {
          while (oldPulses == pulseCount)
            {};
          oldPulses=pulseCount;
          heading = 360 - (unsigned int)(pulseCount+1)*360/pulsesPerRev;
          printf("%d %d\n", heading, speed);
        };
  }



/*---------------------------------------------------------------------*/
/* Set up interrupt vectors                                            */
/*---------------------------------------------------------------------*/
#include "vectors1.c"
```