MicroMouse Design
The Intellimouse Explorer

Date Submitted:
May 13, 2005

Sponsor: Dr. Herb Hess
hhess@uidaho.edu

Advisors: Dr. Herb Hess & Dr. Brian Johnson
hhess@uidaho.edu & bjohnson@uidaho.edu

Viola Fucsko
fucs8472@uidaho.edu

Bijan Houle
houl6492@uidaho.edu

Henis Mitro
mitr5986@uidaho.edu

Mikel Olson
olso7320@uidaho.edu
# Table of Contents

Abstract iii

1. Project Description 1
   1.1 Overview 1
   1.2 Objectives 1
   1.3 Methods 1

2. Status and Preliminary Results 2

3. Methods 6
   3.1 Microcontroller 6
   3.2 Drive Components 7
   3.3 Sensors 8
   3.4 Chassis and Wheels 8

4. Validation Procedure 10

Appendix 12
Abstract

A MicroMouse is being designed to negotiate a path to the center of a maze in the optimal amount of time. The design must integrate a microcontroller, sensors, and motors. This paper presents the current design and interface of the different modules as well as preliminary data related to the performance of the MicroMouse. The motors and sensors have been connected to the microcontroller and currently, the MicroMouse is capable of moving a specified distance in a specified direction upon command, as well as to receive some sensor input and stop before hitting a wall. Future challenges lie in using the sensor input for maze navigation and finding the optimal algorithm for solving the maze.


1. Project Description

1.1 Overview

The goal of this project is to design a self-contained, autonomous, electrically powered vehicle called a “MicroMouse”, to negotiate a path to the center of a maze. The MicroMouse will be built in accordance with the IEEE APEC MicroMouse Contest Rules.

1.2 Objectives

From these rules we have generated the following objectives for our MicroMouse:

- Autonomously find a way to the goal of the maze and back to the starting point.
- Complete a run (from start to goal) in less than ten minutes.
- Turn 360° in a single grid square of the maze.
- The main body must fit inside of a single grid of the maze (16.8 cm x 16.8 cm).
- The sensor wing must fit inside a 25 cm x 25 cm square.

1.3 Methods

To accomplish these objectives we have divided the project into four main components: microcontroller, drive components, sensors, and chassis. We will use a microcontroller, stepper motors and distance and proximity sensors to create our MicroMouse. The MicroMouse will be setup using a two-wheel drive system. Stepper motors will be connected to the drive wheels with ball casters providing balance. Distance sensors will be used to measure the length of the path on the front, left and right sides of the MicroMouse. Knowing the distances to the walls in those directions will allow for more
intelligent maze navigation. Proximity sensors will be located on wings that extend above the maze walls to the sides of the MicroMouse. They will detect the presence of side walls, as well as perpendicular walls in adjacent cells. The microcontroller will keep track of maze information, control the navigation of the mouse, and optimize the path back to the start.

2. Status and Preliminary Results

As illustrated by Figure 1, a majority of the modules described in the previous section are completed and fully operational. A more detailed list of these modules is presented below:

- Basic assembly of all components (See Figure 2.)
- Straight-line and turning movement (See Table 3 in Appendix B.)
  - The Intellimouse is able to move in a straight-line with a deviation of 35.4 cm from the center path while traveling a 285 cm path.
  - The current maximum average speed in the straight-line movement is 0.107 m/s, which is a little below the desired velocity.
  - The system is able to turn 180 degrees within a maze block in a period of 1.25 seconds with an error of just over 1 degree.
  - Current data shows that although the MicroMouse performs a 90-degree turn with a small error, the error does not accumulate during continuous left or right turns.

- Test maze
  - A 6 x 6 test maze has been built with a plywood base and foam walls.
  - The tops of the walls are spray painted with red, while the base is black in accordance with the specifications.
  - The maze walls are adjustable for different configurations.
- A picture of the maze is shown in Figure 3.

![Figure 3: Test Maze](image)

- **Basic sensor to board communication**
  - Sharp GP2D12 distance sensors are interfaced to the microcontroller and sensor output is routed to the Handy Board LCD.
  - The sensors have been tested to obtain a calibration curve of distance to nearest wall vs. voltage and microcontroller output. (See Table 4 and graphs in Appendix B.)
  - Sensor input is used to detect the distance to the front wall and stop before collision.

- **Motor control algorithm**
  - Stepper motors are driven by the microcontroller using full stepping and the built in microcontroller functions. (See code in Appendix D.)
  - Motors can be controlled to move in any specified direction for any specified distance.
• Full stepping results in a noticeable error when turning 90 degrees, however this is corrected by alternating the number of steps between consecutive turns.

• Batteries
  • The MicroMouse is powered from the on board microcontroller batteries. The latest data demonstrated that the batteries are capable of driving the MicroMouse for more than 1 hour and 10 minutes, which eliminates the need for external rechargeable batteries.

Under Test
• Redesign motor algorithm to incorporate half-stepping for increased speed.
  • The present average top speed of the system is 0.107 m/s, and work is being done to increase speed to 0.5 m/s.

• Accurate sensor to board communication
  • Improve data acquisition from sensors.
  • Install Fairchild QRB 1134 sensors on wings.
  • Use sensor data for maze navigation.

• Straight-line movement accuracy
  • Decrease system deviation from center path during straight-line movement.
    • This will be done using feedback from the proximity sensors and will also be improved once a final chassis has been built.

Under Design
• Chassis
  • Robust and compact chassis.
  • Adequate to allow easy movement inside the maze.
• Complete Maze
  • Create a full maze in accordance with IEEE specifications.

3. Methods

Figure 4: Relationship Between MicroMouse Modules

3.1 Microcontroller

The microcontroller is both the brain and the heart of the MicroMouse. Because the device is autonomous, the microcontroller must control everything the MicroMouse does while in the maze. Based on the inputs it receives from the sensors it must calculate a path for the mouse to navigate the maze. It must also control the drive motors to move the mouse and after reaching the goal reduce the path back to the starting square to an optimal path. The microcontroller board must be low power and the entire board must be smaller than 25 x 25cm. For this project a microcontroller that could be programmed in some form of C was also desirable. Based on these requirements we decided to use the Handy Board [4], a Motorola 68HC11-based microprocessor.

The work on the microcontroller can be broken down into different functions: path finding, movement, sensing, and crash recovery. The path finding algorithm will receive inputs from the sensors, calculate the best move, then output commands to the
movement functions. Movement control and sensor data gathering functions will be established early to ensure their proper function and allow ample time for debugging. In order to combat unforeseen skids and collisions, a variety of crash recovery methods will have to be developed. As with any large program, the functions can be worked on simultaneously as long as strict variable passing rules are established before work begins.

3.2 Drive Components

The MicroMouse’s drive system consists of two stepper motors. Turning is controlled by rotating each motor in a different direction, allowing $360^\circ$ turning in a single square. Because of the general design of a maze, acceleration dynamics will play a larger part in the overall maze negotiation time of the mouse than its top speed. Therefore, it is very important that the mouse is able to stop, speed up, and turn quickly. Based on a worst-case analysis of the maze with our time goal of 10 minutes for a run, the motors will need to be able to accelerate the mouse at $1\text{ m/s}^2$ and drive the mouse at an average speed of about $0.1\text{ m/s}$.

To be supplied by a reasonable number of batteries, the motors will need to be rated for about 6-12V at under 1.25A each. Controlling the motors will be simplified by the fact that the Handy Board contains on onboard motor controller and many resources on stepper motor control. While the Handy Board can supply power directly to the motors, it may be necessary to buffer the output of the board to allow the motors to be powered externally.
3.3 Sensors

In the final MicroMouse, two types of sensors will be used. A row of proximity sensors overhanging the top of the maze on both sides of the MicroMouse will be used to keep the MicroMouse centered within the cell and for environment mapping in adjacent cells. Fairchild QRB 1134 proximity sensors have been selected, which are a combination of a light emitting diode and a phototransistor.

Sharp GP2D12 distance sensors will be used to detect the length of the path in front, and possibly to the sides of the MicroMouse. These infrared sensors are relatively low cost, have been used effectively in several MicroMouse designs, and transmit an output related to the distance to the nearest object within a range of 10 to 80 cm. The sensors will have to be calibrated to function predictably for varying maze and ambient conditions. Initial testing has shown that the output of the sensors is not significantly affected by the level of ambient lighting. In addition, minimal variation in the distance output is acceptable, since the exact distance is not as important as the number of cells between the MicroMouse and the next wall. The proximity sensors will be able to detect a wall/no wall interface when the MicroMouse has reached the end of the cell, at which point the measured distance should be an approximate multiple of 18 cm, and the distance output of the Sharp sensors could be appropriately adjusted. However, additional factors that may adversely affect the sensors, such as camera flashes or other interference must be considered.

3.4 Chassis and Wheels

The MicroMouse chassis is the main piece that holds together all the other parts. It must provide enough space for the microprocessor, batteries, motors and sensors. The
chassis will be built with an easy to work material like plastic or aluminum. The advantage of using aluminum would be to draw heat away from the motors, functioning as a built-in heat sink, but plastic may be easier and cheaper.

The chassis need to be small enough so that is able to make a full turn inside a maze block without touching the walls. This will be the first component to be built because it will allow the design team to perform tests on the MicroMouse components. It would be advantageous to use a more compact triangle form, with the two drive wheels directing the MicroMouse and a front or back caster wheel to balance the system. This shape allows for easier design of a mouse that rotates freely.

The MicroMouse will require two wheels with enough traction to avoid slipping, and two caster wheels for balance. The wheel diameter is an important factor to consider. If the diameter is too large, the mouse will tip over more easily; if the diameter is too small, the mouse may drag or produce too little torque. The weight of the wheels will need to be small enough so that they do not add extra load to the motors.

The wheels were selected based on the above specifications. The GMPW wheels are made to match almost any DC or stepper motor. The wheels are molded from ABS, and they measure just less than 65 mm across by 7.62 mm wide. With the included rubber band, the total wheel diameter is 65 mm. The ball caster from Tamiya's educational construction series is a small steel ball that rests on metal rollers inside a plastic frame. The caster can be assembled in two different ways, so that it has a total height of either 25 mm or 35 mm.
4. Validation Procedure

Each module can be developed independently, however many interconnect considerations must be made. The basic flow of the design process can be illustrated as follows:

- Basic Motion
  - Attach the microcontroller to motors. Be able to control the rotation of the motors using the microcontroller board. Develop c code to determine the number of steps required for the motor during straight-line movement and turning, while minimizing off track alignment.

- Straight-line movement
  - Attach the motors and microcontroller board to the chassis. Develop the control system to send the mouse in a straight line. Over the longest possible straight-line distance (2.88m), the deviation can be at maximum 9cm, or 3.125%. The minimum average speed for a specified distance should be 0.1 m/s.

- Turning movement
  - Develop the control system to turn the mouse in place. It must be able to complete both a 90 and 180 turn, within 2 degrees, or 1.11%. The system should be able to make a 360 turn within 3 seconds.

- Max speed and acceleration testing, both straight-line and turning
  - The mouse must be stress-tested to collect data on its peak performance under normal conditions. Knowing the upper bounds on the performance of the
mouse will allow us to set more reasonable nominal values for top speed, turning speed, braking distance, and other characteristics.

- Basic sensor data acquisition, verify correct interpretation of data
  - Attach sensors to chassis and microcontroller. Verify that the sensors are sensing and MicroMouse is reading the correct values under a variety of operating conditions.
    - Verify the Sharp GP2D12 sensor between 10-80 cm with 2.5cm incremental and analyze the calibration curve.
    - Verify the range of Fairchild QRB1134 also, and analyze the data received.

- In-Maze:
  - Controlled straight-line movement and dynamics, using wall detection
    - All components must be connected at this point. Close the straight-line control loop, and verify that the mouse can travel down a straight path, avoiding walls.
  - Controlled turning movement and dynamics, using wall detection
    - Verify that the mouse can detect a wall in front of it, and recognize the need to turn. Verify that it can complete both a 90° and 180° turn successfully.
  - Close the entire control loop and check for correct autonomous function
    - Check behavior in the following situations: straight-line, turn, dead end, center detection, error detection and correction, and general overall performance.
Appendices

Appendix A: Bill of Materials 13
Appendix B: Experimental Data 14
Appendix C: Interface Circuitry for QRB 1134 Sensors 17
Appendix C: Code 18
## Appendix A: Bill of Materials

### Table 1: Current Budget

<table>
<thead>
<tr>
<th>Component</th>
<th>Estimated Amount</th>
<th>Actual Cost</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller (Motorola 68HC11)</td>
<td>$350</td>
<td>$305.00</td>
<td>$45.00</td>
</tr>
<tr>
<td>Stepper Motors</td>
<td>$45</td>
<td>$43.64</td>
<td>$1.36</td>
</tr>
<tr>
<td>Sensors (Sharp GP2D12/Fairchild QRB1134)</td>
<td>$60</td>
<td>$40.90</td>
<td>$19.10</td>
</tr>
<tr>
<td>Mouse Chassis</td>
<td>$20</td>
<td>$0.00</td>
<td>$20.00</td>
</tr>
<tr>
<td>12 Pack NiMH Rechargeable Batteries</td>
<td>$35</td>
<td>$0.00</td>
<td>$35.00</td>
</tr>
<tr>
<td>Battery Charger</td>
<td>$20</td>
<td>$0.00</td>
<td>$20.00</td>
</tr>
<tr>
<td>Maze</td>
<td>$60</td>
<td>$60.28</td>
<td>($0.28)</td>
</tr>
<tr>
<td>Passive Components</td>
<td>$100</td>
<td>$0.00</td>
<td>$100.00</td>
</tr>
<tr>
<td>Active Components</td>
<td>$100</td>
<td>$0.00</td>
<td>$100.00</td>
</tr>
<tr>
<td>Printed Circuit Board</td>
<td>$200</td>
<td>$0.00</td>
<td>$200.00</td>
</tr>
<tr>
<td>Posters &amp; Report Copies</td>
<td>$25</td>
<td>$0.00</td>
<td>$25.00</td>
</tr>
<tr>
<td>Wheels &amp; Ball Casters</td>
<td>$25</td>
<td>$25.00</td>
<td>($25.00)</td>
</tr>
<tr>
<td><strong>Totals</strong></td>
<td><strong>$1,015</strong></td>
<td><strong>$474.82</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Remaining Budget** $540.18

### Table 2: Parts List

<table>
<thead>
<tr>
<th>Component</th>
<th>Part Number</th>
<th>Supplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microprocessor (Motorola 68HC11)</td>
<td>MIT Handyboard</td>
<td><a href="http://www.gleasonresearch.com">www.gleasonresearch.com</a></td>
</tr>
<tr>
<td>Stepper Motors</td>
<td>Reliapro 42BYG205</td>
<td><a href="http://www.jameco.com">www.jameco.com</a></td>
</tr>
<tr>
<td>Sensors</td>
<td>Sharp GP2D12 &amp; Fairchild QRB1134</td>
<td><a href="http://www.junun.org/MarkIII/Store.jsp">www.junun.org/MarkIII/Store.jsp</a></td>
</tr>
<tr>
<td>Wheels</td>
<td>Solarbotics GMPW-R</td>
<td><a href="http://www.hobbyengineering.com">www.hobbyengineering.com</a></td>
</tr>
<tr>
<td>Ball Caster</td>
<td>Tamiya 70144</td>
<td><a href="http://www.robotmarketplace.com">www.robotmarketplace.com</a></td>
</tr>
</tbody>
</table>
Appendix B: Experimental Data:

Motors:

Table 3: Movement Errors

<table>
<thead>
<tr>
<th>Type of Motion and Error Being Measured</th>
<th>Desired Motion</th>
<th>Actual Motion</th>
<th>Amount of Error</th>
<th>Percent Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>straight-line motion, short range</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-straight-line distance travelled (cm)</td>
<td>107.36</td>
<td>107.5</td>
<td>0.14</td>
<td>0.13</td>
</tr>
<tr>
<td>-offset perpendicular to direction of motion (to the left) (cm)</td>
<td>0</td>
<td>0.8</td>
<td>0.8</td>
<td>0.75</td>
</tr>
<tr>
<td>straight-line motion, maximum range</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-straight-line distance travelled (cm)</td>
<td>288</td>
<td>285</td>
<td>3</td>
<td>1.04</td>
</tr>
<tr>
<td>-offset perpendicular to direction of motion (to the left) (cm)</td>
<td>0</td>
<td>35.4</td>
<td>35.4</td>
<td>12.29</td>
</tr>
<tr>
<td>-rotational offset counterclockwise (degrees)</td>
<td>0</td>
<td>20.8</td>
<td>20.6</td>
<td>N/A</td>
</tr>
<tr>
<td>turning motion (degrees)</td>
<td>720</td>
<td>725</td>
<td>5</td>
<td>0.69</td>
</tr>
<tr>
<td>average velocity (m/sec)</td>
<td>at least 0.1</td>
<td>0.107</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>angular velocity (degrees/sec)</td>
<td>at least 120</td>
<td>144</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Sensors:

Table 4: Performance of Sharp GP2D12 Sensors

<table>
<thead>
<tr>
<th>Distance (cm)</th>
<th>Vout (V)</th>
<th>Microcontroller Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.340</td>
<td>53</td>
</tr>
<tr>
<td>5</td>
<td>1.672</td>
<td>89</td>
</tr>
<tr>
<td>10</td>
<td>2.308</td>
<td>119</td>
</tr>
<tr>
<td>15</td>
<td>1.611</td>
<td>85</td>
</tr>
<tr>
<td>20</td>
<td>1.245</td>
<td>65</td>
</tr>
<tr>
<td>25</td>
<td>1.004</td>
<td>53</td>
</tr>
<tr>
<td>30</td>
<td>0.840</td>
<td>44</td>
</tr>
<tr>
<td>35</td>
<td>0.737</td>
<td>38</td>
</tr>
<tr>
<td>40</td>
<td>0.639</td>
<td>33</td>
</tr>
<tr>
<td>45</td>
<td>0.562</td>
<td>30</td>
</tr>
<tr>
<td>50</td>
<td>0.506</td>
<td>28</td>
</tr>
<tr>
<td>55</td>
<td>0.465</td>
<td>24</td>
</tr>
<tr>
<td>60</td>
<td>0.430</td>
<td>21</td>
</tr>
<tr>
<td>65</td>
<td>0.392</td>
<td>19</td>
</tr>
<tr>
<td>70</td>
<td>0.361</td>
<td>18</td>
</tr>
<tr>
<td>75</td>
<td>0.340</td>
<td>16</td>
</tr>
<tr>
<td>80</td>
<td>0.330</td>
<td>14</td>
</tr>
<tr>
<td>85</td>
<td>0.280</td>
<td>4</td>
</tr>
</tbody>
</table>
Appendix C: Interface Circuitry for QRB 1134 Sensors

![Diagram of QRB 1134 sensor interface circuitry]

- +5V
- Pull-up resistor
- ORB1134
- GND
- Anode (orange)
- Collector (white)
- Cathode (green)
- Emitter (blue)
//Motor vectors
//Motor-1
int CCW1[4] = {1, 0, 1, 0};
int CW1[4] = {0, 1, 0, 1};
//Motor-2
int CW2[4] = {2, 3, 2, 3};
int CCW2[4] = {3, 2, 3, 2};

//Indicates the last motor step to allow for locked
wheels
int LastMotor1 = 0;
int LastMotor2 = 2;

//integer for menu navigation
int iCommands = 0;

//Delay between motor steps
#define MOTOR_DELAY 0.001

//Steps for one revolution
#define REV_CYCLES 50

//Steps for 90deg turn
#define TURN_CYCLES 18

int turn_cycles_right = 18; //Global variables, correct later
int turn_cycles_left = 18;

//Activates the specified motor vector for iCycles
void MoveMotor(int iMotorDir[], int iCycles)
{
    int v;
    for(x=0; x<iCycles; x++)
    {
        int y = 0;
        fd(iMotorDir[y]);
        sleep(MOTOR_DELAY);
        if(MotorDir[0] < 2)
            off(LastMotor1);
        else
            off(LastMotor2);
        y++;
    }
    bk(iMotorDir[y]);
    sleep(MOTOR_DELAY);
    off(iMotorDir[y+1]);
    y++;
    bk(iMotorDir[y]);
    sleep(MOTOR_DELAY);
    off(iMotorDir[y-1]);
    if(MotorDir[0] < 2)
    {
        LastMotor1 = MotorDir[y];
    }
    else
    {
        LastMotor2 = MotorDir[y];
    }
}

//Both motors moving in opposite directions
//iDir=0 forward
//iDir=1 backward
void Move(int iDir, int iCycles)
{
    if(iDir == 0)
    {
        start_process(MoveMotor(CW1, iCycles));
        MoveMotor(CW2, iCycles);
        if (iCycles == 18)
            turn_cycles_left = 17;
        else
            turn_cycles_left = 18;
    }
    else
    {
        start_process(MoveMotor(CCW1, iCycles));
        MoveMotor(CCW2, iCycles);
        if (iCycles == 18)
            turn_cycles_right = 17;
        else
            turn_cycles_right = 18;
    }
    //Pivots the mouse about one wheel
    //iDir=0 left
    //iDir=1 right
    void TurnSquare(int iDir, int iCycles)
    {
        if(iDir == 0)
        {
            start_process(MoveMotor(CCW1, iCycles));
            MoveMotor(CCW2, iCycles);
            if (iCycles == 18)
                turn_cycles_left = 17;
            else
                turn_cycles_left = 18;
        }
        else
        {
            start_process(MoveMotor(CCW1, iCycles));
            MoveMotor(CCW2, iCycles);
            if (iCycles == 18)
                turn_cycles_right = 17;
            else
                turn_cycles_right = 18;
        }
    }

    //Code for prototype demonstration
    //Stop cycles commands, start begins indicated
    //command, knob value is distance
    void TestMain()
    {
        while(1)
        {
            if(stop_button())
            {
                Commands++;
                sleep(0.25);
                if(Command == 1)
                    print("Forward\n");
                else if(Command == 2)
                    print("Backward\n");
                else if(Command == 3)
                    print("TurnLeft\n");
                else if(Command == 4)
                    print("TurnRight\n");
                else if(Command == 5)
                    print("SquareRight\n");
                else if(Command == 6)
                    print("SquareLeft\n");
                else if(Command == 7)
                    print("ToWall\n");
                else if(Command == 8)
                    print("Battery\n");
            }
            else
            {
                Commands = 0;
                print("Fwd-Back-180\n");
            }
        }
    }

    //Battery test
    void main()
    {
        int x = 0;
        float y;
        for(;;)
        {
            y = 0.346; //random number
            Move(0, knob());
            Turn(0, turn_cycles_left);
            x = analog(2);
            if(x < 100)
            {
                y = (float)x*0.0234+y;
            }
            else
            {
                Move(0, knob());
                Turn(0, turn_cycles_left);
            }
        }
    }

    //ToWall
    void ToWall()
    {
        int x = 0;
        while(x < 100)
        {
            x = analog(2);
            Move(0, knob());
            Move(0, 5);
        }
    }

    //SquareLeft
    void SquareLeft()
    {
        int x;
        for(x=0; x<4; x++)
        {
            Move(0, knob());
            Turn(0, turn_cycles_left);
        }
    }

    //SquareRight
    void SquareRight()
    {
        int x;
        for(x=0; x<4; x++)
        {
            Move(0, knob());
            Turn(0, turn_cycles_right);
        }
    }

    //Fwd-Back-180
    void Fwd-Back-180()
    {
        int x;
        for(x=0; x<2; x++)
        {
            Move(0, knob());
            Turn(0, 2*TURN_CYCLES-1);
        }
    }

    //Forward
    void Forward()
    {
        int x;
        for(x=0; x<2; x++)
        {
            Move(0, knob());
            Turn(0, turn_cycles_right);
        }
    }

    //Backward
    void Backward()
    {
        int x;
        for(x=0; x<2; x++)
        {
            Move(0, knob());
            Turn(0, turn_cycles_left);
        }
    }

    //Test for
    TestMain();
}