Micromouse Final Project

Andrew Dopson

Oskar Otoya

Spring 2013

**Introduction**

Throughout the entire semester, we have been participating in labs whose primary objectives were to complete small tasks within the micromouse that we could, in the end, put together to help us reach our overall objective. This is where we have put everything together in order to have a micromouse solve an unknown maze by itself. Our primary objective was to develop an algorithm that would enable the micromouse to solve the maze without initially knowing any of the internal wall locations. Using our (working) solutions from the previous labs, we had everything we needed in order to use our algorithm correctly.

**Problem Statement**

We want our micromouse to be able to solve a 6x13 cell maze by going from the bottom left cell to the center of the maze as fast as possible.

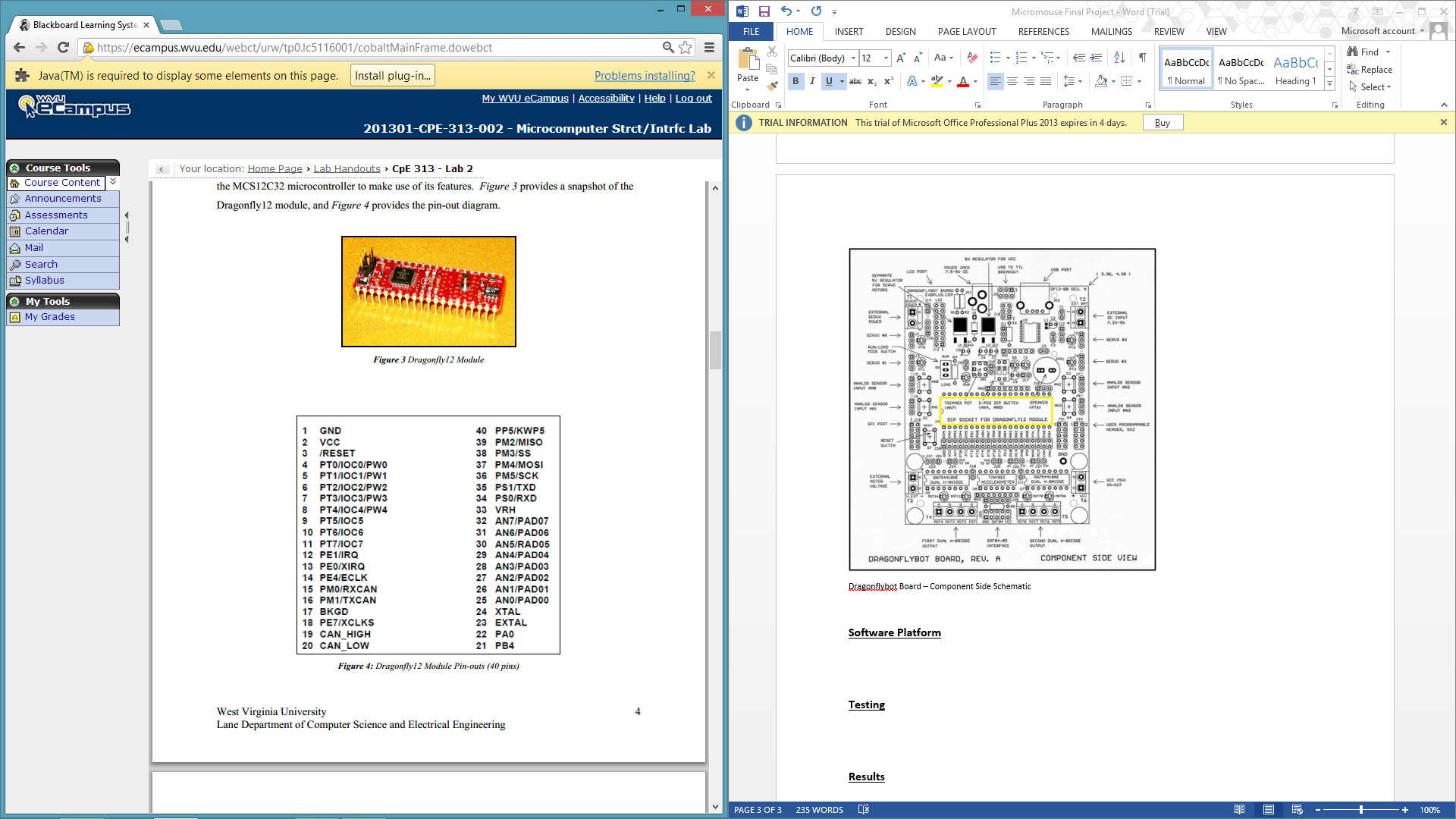
One of the major obstacles that come across is the fact that the micromouse initially does not know any of the wall locations besides the predetermined boundaries. Without knowing the wall locations, the mouse will have no idea of which direction to go.

We will be designing a modified flood fill algorithm programmed to the mouse in order to keep track of its location and update wall locations around it in order to successfully solve the maze.

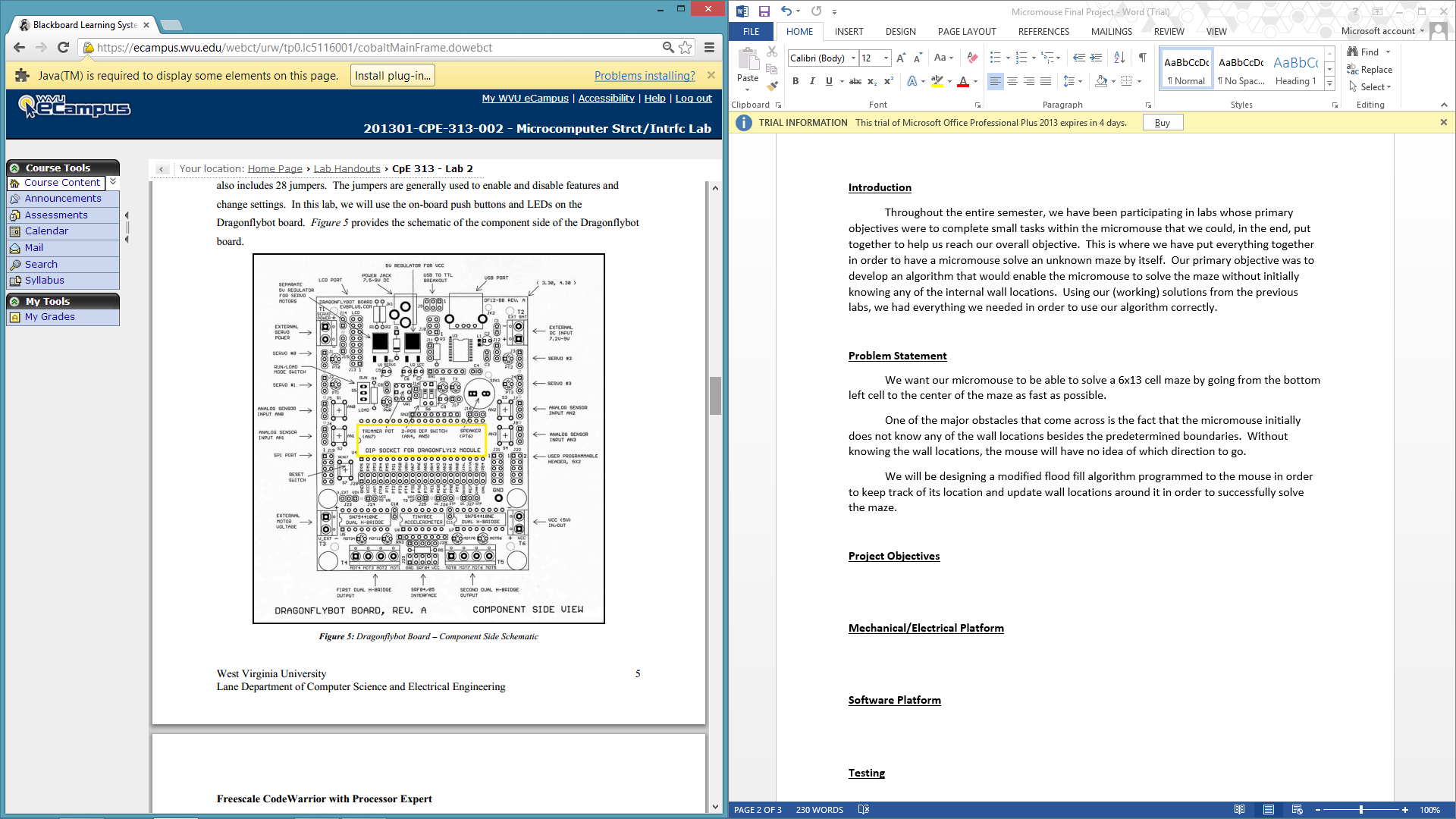
**Project Objectives**

The primary objective of this project was to develop a modified flood fill algorithm that we could program to our micromouse in order to successfully complete an unknown maze. In order to do this we needed to fully understand the modified flood fill algorithm. The maze is made up of a 6x13 matrix. Each cell in the maze is given an initial distance value from the center (final destination) of the maze. Each cell also keeps track of where walls are located around that cell. **We used the micromouse sensors to constantly update the wall values and distance values. Using the cell wall location the mouse is able to identify which location to move to next and which not to even bother with.**

**Mechanical/Electrical Platform**



**Figure 1 : Dragonfly Module Pin-outs (40 pins)**



**Figure 2 : Dragonflybot Board – Component Side Schematic**

The dragonflybot board has numerous advantages to it. It’s relatively simple to program using the C language.

**Software Platform**

All of the beans we used for the final project are shown below in Figure 3.

|  |  |  |
| --- | --- | --- |
| **Component Name** | **Type** | **Use** |
| AD1 | ADC | Analog to Digital Converter |
| Motor1\_Timer2 | TimerInt | Controls left motor |
| Motor2\_Timer2 | TimerInt | Controls right motor |
| PID | TimerInt | Keeps mouse centered |
| TI1 | TimerInt | Used for left state sequence |
| TI2 | TimerInt | Used for right state sequence |
| Cpu | MC9S12C32\_48 | Used for delay |
| MotorR\_Drive | BitsIO | Right motor driver |
| MotorR\_Enable | BitIO | Enable right motor |
| PM0 | BitIO | Left motor driver |
| PM1 | BitIO | Left motor driver |
| PT4 | BitIO | Left motor driver |
| PT5 | BitIO | Left motor driver |

Our program consisted of a basic while loop that continued to run until the mouse reached the center positions. Inside of this while loop were three functions : checkWalls(), changeDistanceVal(), and moveNextCell(). The first method called is the checkWalls method. This uses the current cell the mouse is in and uses the sensor values provided by the pollSensors() method to update the walls in the current cell. If there is also an adjacent cell where a new wall is created, then that cell’s walls are also updated.

The next method called is the changeDistanceVal() method. This method checks the neighboring cells, starting with the north wall and going clockwise. If there is a wall at that neighboring position, that cell is skipped in this method. But if there is no wall, the distance value in that neighboring cell is gotten. If there is more than one open neighboring cell, those values are compared and the smallest value is kept. Then the current cell that the mouse is in is the smallest value plus one. This method also sets the next direction that the mouse will move in next.

Finally the moveNextCell() method is called. This method takes the direction set in the previous method, and moves the micromouse in that direction. This method also takes into account the direction the mouse is currently facing. Also, if the mouse turns left, right, or around, the direction the mouse is facing is updated. Also, depending on which direction the mouse moves to next, the current position will be updated. If it moves to the north/south, the Y value of the position will be incremented/decremented. If it moves east/west, the X position will be incremented/decremented.

Other methods that were used were the turnLeft(), turnRight(), forward(), and around(). These were all involved in the movement of the mouse obviously. This is also where the face direction is updated. Included in the forward method was our PID interrupt. This is enabled while it’s moving and then disabled after it is done. What the PID does is constantly poll the mouse’s sensors. If the mouse is not centered (if the sensors read greater than or less than a certain value) then the timer interrupt period for our motors are then adjusted, making one wheel move a little faster and the other a little slower. Depending on the severity of the mouse being out of alignment, the mouse will make a bigger or smaller adjustment.

Finally, once the mouse reaches the center position, it will exit the while loop and the program will terminate, clarifying that it has solved the unknown maze.

**Testing**

Through the course of building the micromouse we had to perform several tests to ensure our code was running appropriately. In order to accurately iterate through the maze we had to figure out the number of steps needed to move to the next cell, to turn left, turn right, and finally all the way around. So we tested different numbers of steps in the maze to make sure the mouse didn’t run into any walls and would be able to solve the maze without any human interaction.

After dealing with the number of steps we began the testing of our actual code. In order to make sure our entire code worked properly we arranged the maze in different manners. We created a direct path to see if the mouse could even solve the maze, which was the easiest to test. Next we moved on to a little more difficult setup, which involved changing the maze in a different way to see if the mouse could solve it still by making it perform more turns and iteration, but still only staying in the bottom quadrant. Finally we moved to an even more difficult test which incorporated having the mouse go all around the maze to try and solve it. This caused our mouse to do a lot of backtracking. This setup helped us fix our project the most, due to there being more possible problems with a maze that takes a longer route to the center. To check to see if the micromouse knew that it was at the center during the testing we incorporated a method that made the micromouse spin in circles indefinitely until we shut it off this helped to ensure completion.

**Results**

After running some of the tests we became certain that our maze could iterate through the entire maze without touching it under the constraints placed on us as well as through any possible rearrangement that could be thrown at the micromouse. Using the following code our micromouse successfully completed the maze. Our Project.C and Events.C codes are shown below.

/\*\* ###################################################################

\*\* Filename : Project\_2.c

\*\* Project : Project\_2

\*\* Processor : MC9S12C32MFA25

\*\* Version : Driver 01.14

\*\* Compiler : CodeWarrior HC12 C Compiler

\*\* Date/Time : 1/29/2013, 6:43 PM

\*\* Abstract :

\*\* Main module.

\*\* This module contains user's application code.

\*\* Settings :

\*\* Contents :

\*\* No public methods

\*\*

\*\* ###################################################################\*/

/\* MODULE Project\_2 \*/

/\* Including needed modules to compile this module/procedure \*/

#include "Cpu.h"

#include "Events.h"

#include "LCD\_RS.h"

#include "LCD\_RS.h"

#include "LCD\_EN.h"

#include "MotorR\_Drive.h"

#include "AD1.h"

#include "PM0.h"

#include "PM1.h"

#include "PT4.h"

#include "PT5.h"

#include "MotorR\_Enable.h"

#include "PT0.h"

#include "PT1.h"

#include "Motor1\_Timer2.h"

#include "Motor2\_Timer2.h"

#include "TI1.h"

#include "TI2.h"

#include "PID.h"

/\* Include shared modules, which are used for whole project \*/

#include "PE\_Types.h"

#include "PE\_Error.h"

#include "PE\_Const.h"

#include "IO\_Map.h"

#include "math.h"

#define ON 1

#define OFF 0

/\* User includes (#include below this line is not maintained by Processor Expert) \*/

//this outlines the methods that will be used in the program

//this includes global variables used in the program

void writeCom(byte command);

void initializeLCD();

void writeData(byte data);

void clearLCD();

void displaySensors();

void pollSensors();

void convertSensors();

void moveNextCell();

void changeDistanceVal();

void checkWalls();

byte sensors[3];

void Power\_Motors(int power);

void writeBits(int number, byte sequence);

void motorStep();

void turnLeft(int steps);

void forward(int steps);

void turnRight(int steps);

void around(int steps);

extern Motor1\_Timer2\_onInterrupt(void);

extern Motor2\_Timer2\_onInterrupt(void);

extern int errorVal;

int toggle1;

int toggle2;

byte sequence[8]={0x01,0x09,0x08,0x0C,0x04,0x06,0x02,0x03};

int next\_state1=0;

int next\_state2=0;

int number, ret;

int steps1;

int steps2;

int direction1;

int direction2;

byte error;

int period1=4000;

int period2=4000;

int error1;

int gain=5;

//creation of the maze that holds that cell’s wall values

byte wallMaze[13][6] = {{0x09,0x08,0x08,0x08,0x08,0x0C},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x01,0x00,0x00,0x00,0x00,0x04},

{0x07,0x03,0x02,0x02,0x02,0x06}};

//flooding the maze with values

int maze[13][6] = {{8, 7, 6, 6, 7, 8},

{7, 6, 5, 5, 6, 7},

{6, 5, 4, 4, 5, 6},

{5, 4, 3, 3, 4, 5},

{4, 3, 2, 2, 3, 4},

{3, 2, 1, 1, 2, 3},

{2, 1, 0, 0, 1, 2},

{3, 2, 1, 1, 2, 3},

{4, 3, 2, 2, 3, 4},

{5, 4, 3, 3, 4, 5},

{6, 5, 4, 4, 5, 6},

{7, 6, 5, 5, 6, 7},

{8, 7, 6, 6, 7, 8}};

int mousePositionX = 0; //beginning position in the maze

int mousePositionY = 12; //beginning position in the maze

int mouseFace = 0; //used to check direction the mouse is facing

int distanceVal; //temporary variable used to keep distance values

int nextDirection = 0; //temporary variable used to tell which direction to move next

void main(void)

{

/\* Write your local variable definition here \*/

/\*\*\* Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! \*\*\*/

PE\_low\_level\_init();

/\*\*\* End of Processor Expert internal initialization. \*\*\*/

/\* Write your code here \*/

/\*This checks to see if the micromouse is in the center position. If it is not it runs each of the methods until the micromouse is there.

\*/

while(!(((mousePositionX == 2) && (mousePositionY == 6))||((mousePositionX == 3) && (mousePositionY == 6)))){

checkWalls();

changeDistanceVal();

moveNextCell();

}

/\*\*\* Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! \*\*\*/

for(;;){}

/\*\*\* Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! \*\*\*/

} /\*\*\* End of main routine. DO NOT MODIFY THIS TEXT!!! \*\*\*/

/\*This method checks the walls around the micromouse using the sensors to determine if there is a wall or not and if there is then we update the wallMaze.

\*/

void checkWalls(){

pollSensors();

//Mouse facing north

if(mouseFace == 0){

//wall to the west

if(sensors[0] > 60){

//updates wall to the west

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x01;

//if there is a cell to west then change east wall value of the cell to the west

if(mousePositionX > 0){

wallMaze[mousePositionY][mousePositionX - 1]= wallMaze[mousePositionY][mousePositionX - 1] | 0x04;

}

}

//wall to the north

if(sensors[1] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x08;

if(mousePositionY > 0){

wallMaze[mousePositionY - 1][mousePositionX]= wallMaze[mousePositionY - 1][mousePositionX] | 0x02;

}

}

//wall to the east

if(sensors[2] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x04;

if(mousePositionX < 5){

wallMaze[mousePositionY][mousePositionX + 1]= wallMaze[mousePositionY][mousePositionX + 1] | 0x01;

}

}

//Mouse facing East

}else if(mouseFace == 1){

//wall to the north

if(sensors[0] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x08;

if(mousePositionY > 0){

wallMaze[mousePositionY - 1][mousePositionX]= wallMaze[mousePositionY - 1][mousePositionX] | 0x02;

}

}

//wall to the east

if(sensors[1] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x04;

if(mousePositionX < 5){

wallMaze[mousePositionY][mousePositionX + 1]= wallMaze[mousePositionY][mousePositionX + 1] | 0x01;

}

}

//wall to the south

if(sensors[2] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x02;

if(mousePositionY < 12){

wallMaze[mousePositionY + 1][mousePositionX]= wallMaze[mousePositionY + 1][mousePositionX] | 0x08;

}

}

}

//Mouse facing South

else if(mouseFace == 2){

//wall to the east

if(sensors[0] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x04;

if(mousePositionX < 5){

wallMaze[mousePositionY][mousePositionX + 1]= wallMaze[mousePositionY][mousePositionX + 1] | 0x01;

}

}

//wall to the south

if(sensors[1] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x02;

if(mousePositionY < 12){

wallMaze[mousePositionY + 1][mousePositionX]= wallMaze[mousePositionY + 1][mousePositionX] | 0x08;

}

}

//wall to the west

if(sensors[2] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x01;

if(mousePositionX > 0){

wallMaze[mousePositionY][mousePositionX - 1]= wallMaze[mousePositionY][mousePositionX - 1] | 0x04;

}

}

}

//Mouse facing West

else if(mouseFace == 3){

//wall to the south

if(sensors[0] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x02;

if(mousePositionY < 12){

wallMaze[mousePositionY + 1][mousePositionX]= wallMaze[mousePositionY + 1][mousePositionX] | 0x08;

}

}

//wall to the west

if(sensors[1] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x01;

if(mousePositionX > 0){

wallMaze[mousePositionY][mousePositionX - 1]= wallMaze[mousePositionY][mousePositionX - 1] | 0x04;

}

}

//wall to the north

if(sensors[2] > 60){

wallMaze[mousePositionY][mousePositionX]= wallMaze[mousePositionY][mousePositionX] | 0x08;

if(mousePositionY > 0){

wallMaze[mousePositionY - 1][mousePositionX]= wallMaze[mousePositionY - 1][mousePositionX] | 0x02;

}

}

}

}

/\* This method checks the neighboring cells, starting with the north wall and going clockwise. If there is a wall at that neighboring position, that cell is skipped in this method. But if there is no wall, the distance value in that neighboring cell is gotten. If there is more than one open neighboring cell, those values are compared and the smallest value is kept. Then the current cell that the mouse is in is the smallest value plus one.

\*/

void changeDistanceVal(){

//if there is no north wall

if((wallMaze[mousePositionY][mousePositionX] & 0x08) == 0){

//changes the distance value and sets the direction to move forward

distanceVal = maze[mousePositionY - 1][mousePositionX];

nextDirection = 0;

}

//if there is no east wall

if((wallMaze[mousePositionY][mousePositionX] & 0x04) == 0){

//if there is a north wall

if((wallMaze[mousePositionY][mousePositionX] & 0x08) != 0){

distanceVal = maze[mousePositionY][mousePositionX + 1];

nextDirection = 1;

} else {

if(distanceVal > maze[mousePositionY][mousePositionX + 1]){

distanceVal = maze[mousePositionY][mousePositionX + 1];

nextDirection = 1;

}

}

}

//if there is no south wall

if((wallMaze[mousePositionY][mousePositionX] & 0x02) == 0){

//if there is a north and east wall

if(((wallMaze[mousePositionY][mousePositionX] & 0x08) != 0) && ((wallMaze[mousePositionY][mousePositionX] & 0x04) != 0)){

distanceVal = maze[mousePositionY + 1][mousePositionX];

nextDirection = 2;

} else {

if(distanceVal > maze[mousePositionY + 1][mousePositionX]){

distanceVal = maze[mousePositionY + 1][mousePositionX];

nextDirection = 2;

}

}

}

//if there is no west wall

if((wallMaze[mousePositionY][mousePositionX] & 0x01) == 0){

if(((wallMaze[mousePositionY][mousePositionX] & 0x08) != 0) && ((wallMaze[mousePositionY][mousePositionX] & 0x04) != 0) && ((wallMaze[mousePositionY][mousePositionX] & 0x02) != 0)){

distanceVal = maze[mousePositionY][mousePositionX - 1];

nextDirection = 3;

} else {

if(distanceVal > maze[mousePositionY][mousePositionX - 1]){

distanceVal = maze[mousePositionY][mousePositionX - 1];

nextDirection = 3;

}

}

}

//reset the value of our current location

maze[mousePositionY][mousePositionX] = distanceVal + 1;

//used a delay to see the mouse move from cell to cell and check its

//iteration

Cpu\_Delay100US(10000);

}

/\*

This method moves the micromouse to the next cell based on the direction we told it to go in and the way the micromouse is facing. After moving the mouse, its position is then updated depending on which direction it travels.

\*/

void moveNextCell(){

//initializeLCD();

//clearLCD();

//displaySensors();

//mouse needs to go north

if(nextDirection == 0){

//mouse is already facing north just move forward

if(mouseFace == 0){

forward(415);

//mouse is facing east turns left 90 degrees and moves forward

}else if(mouseFace == 1){

turnLeft(230);

forward(415);

//mouse is facing south turns 180 and then moves forward

}else if(mouseFace == 2){

around(460);

forward(415);

//mouse if facing west turns 90 degrees and then moves forward

}else if(mouseFace == 3){

turnRight(230);

forward(415);

}

//we reset the mouseFace to face north since we just moved north

//also changed the mousePosition

mouseFace = 0;

mousePositionY--;

//mouse needs to move to the east

}else if(nextDirection == 1){

if(mouseFace == 0){

turnRight(230);

forward(415);

}else if(mouseFace == 1){

forward(415);

}else if(mouseFace == 2){

turnLeft(230);

forward(415);

}else if(mouseFace == 3){

around(460);

forward(415);

}

mouseFace = 1;

mousePositionX++;

//mouse needs to move back to the south

}else if(nextDirection == 2){

if(mouseFace == 0){

around(460);

forward(415);

}else if(mouseFace == 1){

turnRight(230);

forward(415);

}else if(mouseFace == 2){

forward(415);

}else if(mouseFace == 3){

turnLeft(230);

forward(415);

}

mouseFace = 2;

mousePositionY++;

//mouse needs to move to the west

}else if(nextDirection == 3){

if(mouseFace == 0){

turnLeft(230);

forward(415);

}else if(mouseFace == 1){

around(460);

forward(415);

}else if(mouseFace == 2){

turnRight(230);

forward(415);

}else if(mouseFace == 3){

forward(415);

}

mouseFace = 3;

mousePositionX--;

}

}

void writeCom(byte command) {

// Set up transmission

LCD\_EN\_PutVal(0);

LCD\_RS\_PutVal(0);

// Sending the upper nibble

MotorR\_Drive\_PutVal(command >> 4);

LCD\_EN\_PutVal(1);

LCD\_EN\_PutVal(0);

// Sending the lower nibble

MotorR\_Drive\_PutVal(command);

LCD\_EN\_PutVal(1);

LCD\_EN\_PutVal(0);

// Delay

Cpu\_Delay100US(500);

}

void initializeLCD() {

//routine for initializing LCD

Cpu\_Delay100US(160);

writeCom(0x30);

Cpu\_Delay100US(50);

writeCom(0x30);

Cpu\_Delay100US(2);

writeCom(0x30);

writeCom(0x20);

writeCom(0x28);

writeCom(0x0C);

writeCom(0x01);

writeCom(0x06);

Cpu\_Delay100US(100);

}

void writeData(byte data) {

// Set up transmission

LCD\_EN\_PutVal(0);

LCD\_RS\_PutVal(1);

// Sending the upper nibble

MotorR\_Drive\_PutVal(data >> 4);

LCD\_EN\_PutVal(1);

LCD\_EN\_PutVal(0);

// Sending the lower nibble

MotorR\_Drive\_PutVal(data);

LCD\_EN\_PutVal(1);

LCD\_EN\_PutVal(0);

// Delay

Cpu\_Delay100US(500);

}

void clearLCD() {

//command for clearing the display

writeCom(0x01);

}

void displaySensors() {

//splits first number into individual digits and displays them

//writeData((sensors[0]/100)+48);

// writeData(((sensors[0]%100)/10) + 48);

// writeData(((sensors[0]%100)%10) + 48);

// writeData(32);

// writeData(32);

// writeData((sensors[2]/100)+48);

// writeData(((sensors[2]%100)/10) + 48);

// writeData(((sensors[2]%100)%10) + 48);

//commad for carriage return

// writeCom(0xC0);

//splits second number into individual digits and displays them

// writeData(32);

// writeData(32);

// writeData(32);

// writeData((sensors[1]/100)+48);

// writeData(((sensors[1]%100)/10) + 48);

// writeData(((sensors[1]%100)%10) + 48);

writeData(mousePositionX+48);

writeData(32);

writeData(32);

writeData(32);

writeData(mousePositionY+48);

}

void pollSensors() {

AD1\_Measure(1);

AD1\_GetValue(sensors);

}

void convertSensors(){

sensors[0] = 59.0435 \* (powf(0.9683, sensors[0]));

sensors[1] = 59.0435 \* (powf(0.9683, sensors[1]));

sensors[2] = 59.0435 \* (powf(0.9683, sensors[2]));

}

/\*powers on motors by taking in an integer value

(ON or OFF)\*/

void Power\_Motors(int power) {

//if power is off, disable motors

if (power==OFF) {

MotorR\_Enable\_PutVal(0);

PT0\_PutVal(0);

PT1\_PutVal(0);

}

//if power is on, enable motors

if (power==ON) {

MotorR\_Enable\_PutVal(1);

PT0\_PutVal(1);

PT1\_PutVal(1);

}

}

void writeBits(int number, byte sequence){

if(number==1) {

PT5\_PutVal((sequence & 0x01) &&0x01);

PT4\_PutVal((sequence & 0x02) &&0x02);

PM1\_PutVal((sequence & 0x04) &&0x04);

PM0\_PutVal((sequence & 0x08) &&0x08);

} else{

MotorR\_Drive\_PutVal(sequence);

}

}

//Tracks motor steps, Direction, and current state in array

void motorStep(int steps) {

int i=0;

//executes loop for 'steps' amount of times

for(i=0;i<steps;i++) {

//next state for counter-clockwise

if(next\_state2==0){

next\_state2=7;

} else{

next\_state2--;

}

//next state for clockwise

if(next\_state1==7){

next\_state1=0;

} else{

next\_state1++;

}

Cpu\_Delay100US(40);

}

}

//sets motors in same direction so mouse moves forward

void forward(int step) {

Power\_Motors(ON);

steps1 = step;

steps2 = step;

direction1=1;

direction2=1;

PID\_Enable();

//loop that sits until all steps have executed

while( steps1>0 && steps2>0) {

pollSensors();

}

PID\_Disable();

Power\_Motors(OFF);

}

//sets left motor counterclockwise and right motor clockwise

void turnLeft(int step) {

Power\_Motors(ON);

if(mouseFace == 3){

mouseFace = 0;

}else{

mouseFace++;

}

steps1 = step;

steps2 = step;

direction1=1;

direction2=0;

//loop that sits until all steps have executed

while( steps1>0 &&steps2>0) {}

Power\_Motors(OFF);

}

//sets left motor clockwise and right motor counterclockwise

void turnRight(int step) {

Power\_Motors(ON);

if(mouseFace == 0){

mouseFace = 3;

}else{

mouseFace--;

}

steps1 = step;

steps2 = step;

direction1=0;

direction2=1;

//loop that sits until all steps have executed

while( steps1>0 && steps2>0) {}

Power\_Motors(OFF);

}

//same as turnRight but executes for twice the amount of steps

void around(int step) {

Power\_Motors(ON);

if(mouseFace == 0){

mouseFace = 3;

}else{

mouseFace--;

}

if(mouseFace == 0){

mouseFace = 3;

}else{

mouseFace--;

}

steps1 = step;

steps2 = step;

direction1=1;

direction2=0;

//loop that sits until all steps have executed

while( steps1>0 && steps2>0) {}

Power\_Motors(OFF);

}

/\* END Project\_2 \*/

/\*

\*\* ###################################################################

\*\*

\*\* This file was created by Processor Expert 3.02 [04.44]

\*\* for the Freescale HCS12 series of microcontrollers.

\*\*

\*\* ###################################################################

\*/

/\*\* ###################################################################

\*\* Filename : Events.c

\*\* Project : Project\_2

\*\* Processor : MC9S12C32MFA25

\*\* Component : Events

\*\* Version : Driver 01.04

\*\* Compiler : CodeWarrior HC12 C Compiler

\*\* Date/Time : 1/29/2013, 6:43 PM

\*\* Abstract :

\*\* This is user's event module.

\*\* Put your event handler code here.

\*\* Settings :

\*\* Contents :

\*\* No public methods

\*\*

\*\* ###################################################################\*/

/\* MODULE Events \*/

#include "Cpu.h"

#include "Events.h"

/\* User includes (#include below this line is not maintained by Processor Expert) \*/

extern int number;

extern int direction1;

extern int direction2;

extern int toggle1;

extern int toggle2;

extern byte sequence[8];

extern void writeBits(int number, byte sequence);

extern int next\_state1;

extern int next\_state2;

extern int steps1;

extern int steps2;

void Motor2\_Timer2\_OnInterrupt();

void Motor1\_Timer2\_OnInterrupt();

void TI2\_OnInterrupt();

void TI1\_OnInterrupt();

void PID\_OnInterrupt();

extern byte error;

extern void pollSensor();

extern int error1;

extern int period1;

extern int period2;

extern int gain;

extern byte sensors[3];

int errorVal;

#pragma CODE\_SEG DEFAULT

/\*

\*\* ===================================================================

\*\* Event : AD1\_OnEnd (module Events)

\*\*

\*\* Component : AD1 [ADC]

\*\* Description :

\*\* This event is called after the measurement (which consists

\*\* of <1 or more conversions>) is/are finished.

\*\* The event is available only when the <Interrupt

\*\* service/event> property is enabled.

\*\* Parameters : None

\*\* Returns : Nothing

\*\* ===================================================================

\*/

void AD1\_OnEnd(void)

{

/\* Write your code here ... \*/

}

/\*

\*\* ===================================================================

\*\* Event : TI2\_OnInterrupt (module Events)

\*\*

\*\* Component : TI2 [TimerInt]

\*\* Description :

\*\* When a timer interrupt occurs this event is called (only

\*\* when the component is enabled - <Enable> and the events are

\*\* enabled - <EnableEvent>). This event is enabled only if a

\*\* <interrupt service/event> is enabled.

\*\* Parameters : None

\*\* Returns : Nothing

\*\* ===================================================================

\*/

void TI2\_OnInterrupt(void)

{

//makes sure there are still steps to make

if (steps2 >0) {

/\*if statements below tell the motor to go clockwise

or counter-clockwise based on value of 'direction' \*/

if (direction2==0){

if(next\_state2<=0){

next\_state2=7;

} else{

next\_state2--;

}

} if(direction2==1){

if(next\_state2>=7){

next\_state2=0;

} else{

next\_state2++;

}

}

//decrements the number of steps

steps2--;

}

}

/\*

\*\* ===================================================================

\*\* Event : TI1\_OnInterrupt (module Events)

\*\*

\*\* Component : TI1 [TimerInt]

\*\* Description :

\*\* When a timer interrupt occurs this event is called (only

\*\* when the component is enabled - <Enable> and the events are

\*\* enabled - <EnableEvent>). This event is enabled only if a

\*\* <interrupt service/event> is enabled.

\*\* Parameters : None

\*\* Returns : Nothing

\*\* ===================================================================

\*/

void TI1\_OnInterrupt(void)

{

//makes sure there are still steps to make

if (steps1 >0) {

/\*if statements below tell the motor to go clockwise

or counter-clockwise based on value of 'direction' \*/

if (direction1==0){

if(next\_state1<=0){

next\_state1=7;

}

else{

next\_state1--;

}

}

if(direction1==1){

if(next\_state1>=7){

next\_state1=0;

}

else{

next\_state1++;

}

}

//decrements the number of steps

steps1--;

}

}

/\*

\*\* ===================================================================

\*\* Event : Motor2\_Timer2\_OnInterrupt (module Events)

\*\*

\*\* Component : Motor2\_Timer2 [TimerInt]

\*\* Description :

\*\* When a timer interrupt occurs this event is called (only

\*\* when the component is enabled - <Enable> and the events are

\*\* enabled - <EnableEvent>). This event is enabled only if a

\*\* <interrupt service/event> is enabled.

\*\* Parameters : None

\*\* Returns : Nothing

\*\* ===================================================================

\*/

//controls the right motor

void Motor2\_Timer2\_OnInterrupt(void)

{

//sets number = 2 so that writeBits writes to the right motor

number=2;

/\*if statement that toggles and then writes the next

state of the sequence to the motor\*/

if (toggle2) {

writeBits(number,0);

toggle2=0;

}

else{

number=2;

writeBits(number,sequence[next\_state1]);

toggle2=1;

}

}

/\*

\*\* ===================================================================

\*\* Event : Motor1\_Timer2\_OnInterrupt (module Events)

\*\*

\*\* Component : Motor1\_Timer2 [TimerInt]

\*\* Description :

\*\* When a timer interrupt occurs this event is called (only

\*\* when the component is enabled - <Enable> and the events are

\*\* enabled - <EnableEvent>). This event is enabled only if a

\*\* <interrupt service/event> is enabled.

\*\* Parameters : None

\*\* Returns : Nothing

\*\* ===================================================================

\*/

//controls the left motor

void Motor1\_Timer2\_OnInterrupt(void)

{

//sets number = 1 so that writeBits writes to the left motor

number=1;

/\*if statement that toggles and then writes the next

state of the sequence to the motor\*/

if (toggle1) {

writeBits(number,0);

toggle1=0;

}

else{

number=1;

writeBits(number,sequence[next\_state2]);

toggle1=1;

}

}

/\*

\*\* ===================================================================

\*\* Event : PID\_OnInterrupt (module Events)

\*\*

\*\* Component : PID [TimerInt]

\*\* Description :

\*\* When a timer interrupt occurs this event is called (only

\*\* when the component is enabled - <Enable> and the events are

\*\* enabled - <EnableEvent>). This event is enabled only if a

\*\* <interrupt service/event> is enabled.

\*\* Parameters : None

\*\* Returns : Nothing

\*\* ===================================================================

\*/

void PID\_OnInterrupt(void)

{

//checks if there is only a left wall

if(sensors[0]>60 && sensors[2]<40){

errorVal=(sensors[0]-125)\*20;

TI1\_SetPeriodUS(4000 + errorVal);

TI2\_SetPeriodUS(4000 - errorVal);

}

//checks if there is only a right wall

if(sensors[2]>40 && sensors[0]<60) {

errorVal=(110-sensors[2])\*20;

TI1\_SetPeriodUS(4000 + errorVal);

TI2\_SetPeriodUS(4000 - errorVal);

}

//checks if there is a right wall and a left wall

if( sensors[0]>60 && sensors[2]>40){

errorVal=(sensors[0]-125)\*20;

TI1\_SetPeriodUS(4000 + errorVal);

TI2\_SetPeriodUS(4000 - errorVal);

}

//checks if there are no walls

if(sensors[0]<60 && sensors[2]<40){

TI1\_SetPeriodUS(4000);

TI2\_SetPeriodUS(4000);

}

}

/\* END Events \*/

/\*

\*\* ###################################################################

\*\*

\*\* This file was created by Processor Expert 3.02 [04.44]

\*\* for the Freescale HCS12 series of microcontrollers.

\*\*

\*\* ###################################################################

\*/

**Conclusion**

Overall this project was successful and we were able to understand the modified flood fill algorithm in order to successfully traverse the maze with our micromouse. Once we started working on the code, it became relatively easy to understand how to solve the maze. The main problem that occurred for us was getting our PID to work correctly, which we later found out was due to our other timer interrupts that controlled the state changes of the wheel motors. Through a bit of tinkering with the code after running some of the tests, we were able to make the micromouse iterate through the whole maze without any problem. I would highly recommend repeating this final project in future labs.