```
 1 /*******************************************
 2 *Joshua Philpott & Michael Goltz
 3 *final_bot_code.c
 4 *
 5 *cleaned up version of the code which ran on the bot
 6 *for the demonstration.
 7 *
 8 *This code contains the contents of floodfill.c
 9 *We had link issues with the compiler when trying to import
10 *the code from a seperate file so we directly added the
11 *floodfill.c code as a temporary fix (which for the scope of
12 *this class has become permenant!).
13 * ******************************************/
14
15 #include <hidef.h>      // common defines and macros
16 #include "derivative.h"   //    derivative-specific definitions
17 #ifndef FLOODFILL_H
18 #define FLOODFILL_H
19 #include "floodfill.h"
20 #endif
21
22 void change_buzzer(int dout);
23 void setup_buzzer(void);
24 void InitializeATD(void);
25 void SetupMDCU(void);
26 void SetupRTI(void);
27 void SetupDDR_Ports(void);
28 void NEXT_STATE(void);
29 void SWITCH_DIRECTION(void);
30
31 void HALF_STEP_FORWARD(int steps);    // move forward x number of steps
32 void RIGHT_STEP_FORWARD(void);
33 void LEFT_STEP_FORWARD(void);
34
35
36 void STATIONARY_TURN(int steps, int direction); // stationary turn calls subfunctions
37 void RAMP_BOTH_DOWN(void);
38 void RAMP_BOTH_UP_REV(int steps, int direction);  // stationary turn
39 void HALF_STEP_REV(int steps, int direction);  // stationary turn
40 void RAMP_BOTH_DOWN_REV(int steps, int direction);  // stationary turn
41 void RAMP_BOTH_UP(void);
42 void ROLLING_TURN(int steps, int direction);
43
44
45 void PIVOT_TURN(int steps, int direction);  // pivot turn calls subfunctions
46 void RAMP_ONE_UP(int steps, int direction);  // pivot turn
47 void RAMP_ONE_DOWN(int steps, int direction);  // pivot turn
48
49
50 volatile double Output_right[2] = { 0 };   //last two distance readings
51 volatile double Output_left[2] = { 0 };   //last two distance readings
52 volatile unsigned char OutputIndex = 0;    //points to the most recent distance Output
   location
53
54 //motor wheel half steps
55 volatile unsigned char RightStepArray[] ={0xC0, 0x80, 0x90, 0x10, 0x30, 0x20, 0x60, 0x40};
56 volatile unsigned char LeftStepArray[] ={0x0C, 0x04, 0x06, 0x02, 0x03, 0x01, 0x09, 0x08};
57 unsigned char SEG7_DISPLAY[10] = {0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};
58                                                          //0       1       2       3
   4     5       6       7       8       9
59 // turn control
60 volatile int G_Block_Step = 520; // steps for length of a block
```

```
 61 volatile int G_90_Turn = 208; // steps for 90 degree turn    208
 62 volatile int G_Current_Orient = 0; //0=N, 1=E, 2=S, 3=W
 63 volatile int G_Turn = 0;  //0=N, 1=E, 2=S,  3=W
 64 volatile int prev_explore = -1;
 65 volatile int Temp_New_Direction = 0;
 66 volatile int G_Steps = 0;
 67 volatile int state = 0;
 68 volatile int direction = 0;    // 0 = left, 1 = right
 69
 70
 71 // motor timing global variable
 72 volatile long a[2] =  {-25575, 10508};
 73 volatile long b[3] =  {329, 658, 329};
 74 volatile double kp = 0.7; // 1.0
 75 volatile double kd = 0.2;  // 0.9
 76 volatile double error_r = 0;        // represents our distance from desired location
 77 volatile double errorDot_r = 0;    // represents our change in distance from last reading
 78 volatile double error_l = 0;        // represents our distance from desired location
 79 volatile double errorDot_l = 0;    // represents our change in distance from last reading
 80 volatile int diff = 0;
 81 volatile int interruptcount_filter = 0;
 82 volatile int interruptcount_left = 0;
 83 volatile int interruptcount_right = 0;
 84 volatile int stepindex_left = 0;
 85 volatile int stepindex_right = 0;
 86 volatile double speed_left = 26;       // int  26
 87 volatile double speed_right = 26;     // int  26
 88 volatile double top_speed = 26;      // 26
 89 volatile double current_speed = 130;
 90 volatile double bottom_speed = 130;
 91 volatile int speed_counter = 0;
 92 volatile int count_both = 0;
 93 volatile int count_one = 0;
 94
 95
 96 // ATD conversion
 97 volatile long x_s, rx, lx;
 98 volatile int y_s, ry, ly;
 99 volatile long ylng, rylng, lylng;
100 volatile int d[3] = {32};
101 volatile int rd[3] ={32};
102 volatile int ld[3] ={32};
103 volatile long dlng, rdlng, ldlng;
104
105 // maze solving
106 //volatile int visited[3][3] =
107 volatile int Current_x = 0;
108 volatile int Current_y = 0;
109 volatile int G_Finished = 1;
110 volatile short L_wall = 1;
111 volatile short R_wall = 1;
112 volatile short turn = 0;
113 volatile short rolling_flag = 0;
114
115 /* * * * *  * * * *  * * * ** * * * * * * * * * * * * * * * *
116
117     start of maze solving algorithm
118 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
119
120 /* * * Start floodfill.c * * */
121 /* floodfill.c contains the maze solving capability. The
122    drive functions get next move via the explore function
```

```
123    in floodfill.c*/
124
125 volatile int visited[MAZE_WIDTH][MAZE_HEIGHT] = {{0}}; //array representing visited blocks
    in maze
126 volatile int v_x = 0; // next position to visit when going to unvisited positions
127 volatile int v_y = 0;
128
129 volatile int fnum[MAZE_WIDTH][MAZE_HEIGHT]; //numbers assigned from floodfill
130 //0-no walls, 1-north wall, 2-east wall 3- north and east wall
131 volatile int walls[7][7] ={{2,0,0,0,0,0,0}, //(0,0).......(6,0)
132                            {0,0,0,0,0,0,0}, //
133                            {0,0,0,0,0,0,0}, //
134                            {0,0,0,0,0,0,0}, //
135                            {0,0,0,0,0,0,0}, //
136                            {0,0,0,0,0,0,0}, //
137                            {0,0,0,0,0,0,0}};//(0,6).......(6,6)
138
139
140 /* * * * *  * * * *  * * * * ** * * * * * * * * * * * * * * * *
141     increment part of explore
142 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
143 int increment_v(int x, int y){
144 /*
145  * set v_y, v_x to new explore position.
146  *
147  * return 1 if everything is visited
148  * return 0 if still exploring
149  *
150  */
151     //Find next v with lowest floodnumber
152     int vcell=0;
153     int vcell_n=0;
154     int vcell_e=0;
155     int vcell_s=0;
156     int vcell_w=0;
157     int i = 0;
158     int j = 0;
159     int flag = 0;
160     int low_x = 0;
161     int low_y = 0;
162     int low_f = (MAZE_HEIGHT*MAZE_WIDTH)+1;
163
164     initMaze();
165     floodfill(x,y,0);
166
167     for(i=0;i<MAZE_WIDTH;i++){
168         for(j=0;j<MAZE_HEIGHT;j++){
169
170             //j will be x,
171             //i will be y
172             vcell = visited[i][j];
173             if(i < (MAZE_HEIGHT-1)) vcell_n = visited[i+1][j];
174             else vcell_n = 1;
175
176             if(j < (MAZE_WIDTH-1)) vcell_e = visited[i][j+1];
177             else vcell_e =  1;
178
179             if(i>0)vcell_s = visited[i-1][j];
180             else vcell_s = 1;
181
182             if(j>0) vcell_w = visited[i][j-1];
183             else vcell_w = 1;
```

```
184
185             if(vcell_n && vcell_e && vcell_s && vcell_w){
186                 visited[i][j]=1;
187                 vcell = 1;
188             }
189
190
191             if(!vcell){
192                 flag = 1;
193                 if(getFNum(j,i)<low_f){
194                     low_f = getFNum(j,i);
195                     low_x = j;
196                     low_y = i;
197                 }
198             }
199         }
200     }
201
202     if(flag){
203         v_x = low_x;
204         v_y = low_y;
205         return 0;
206     }else{
207         return 1;
208     }
209 }
210 void initMaze(){
211     int i, j = 0;
212     for(i=0;i<MAZE_WIDTH;i++){
213         for(j=0;j<MAZE_HEIGHT; j++){
214             fnum[i][j]=(MAZE_HEIGHT*MAZE_WIDTH)+1;
215         }
216     }
217 }
218 void printMaze(){
219     int i = 0;
220     int j = 0;
221     //printf("\n\n");
222     for(i=MAZE_HEIGHT-1; i>=0; i--){
223         for(j=0; j<MAZE_WIDTH; j++){
224             //printf(" %2d ", fnum[i][j]);
225         }
226             //printf("\n");
227     }
228 }
229 bool isWallWest(int x, int y){
230     if(x==0){
231         return true;
232     }
233     else if(walls[y][x-1]==2 || walls[y][x-1]==3){
234         return true;
235     }
236     else return false;
237
238 bool isWallEast(int x, int y){
239     if(x==MAZE_WIDTH-1){
240         return true;
241     }
242     else if(walls[y][x]==2 || walls[y][x]==3){
243         return true;
244     }
245     else return false;
```

```
246 }
247 bool isWallNorth(int x, int y){
248     if(y==MAZE_HEIGHT-1){
249         return true;
250     }
251     else if(walls[y][x]==1 || walls[y][x]==3){
252         return true;
253     }
254     else return false;
255 }
256 bool isWallSouth(int x, int y){
257     if(y==0){
258         return true;
259     }
260     else if(walls[y-1][x]==1 || walls[y-1][x]==3){
261         return true;
262     }
263     else return false;
264 }
265
266 /* * * * *  * * * *  * * * ** * * * * * * * * * * * * * * *
267     Explore the maze and goal seek
268 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
269 int explore(int x, int y, int goal_x, int goal_y, int orientation, int front, int left, int
    right){
270     /*****************************************
271     *explore
272     *
273     *returns
274     *   -1      - error
275     *   0 - 3   - direction (N,E,W,S)
276     *   4       - maze complete
277     * ***************************************/
278     int cell_f_num = (MAZE_HEIGHT*MAZE_WIDTH)+1;
279     int temp = (MAZE_HEIGHT*MAZE_WIDTH)+1;
280     int f_num_dif = (MAZE_HEIGHT*MAZE_WIDTH)+1;
281     int dir = -1;
282     static int found_goal=0;
283     static int maze_explore_complete=0;
284     static int maze_return_start=0;
285     if(!maze_explore_complete)mapCell(x, y, orientation, front, left, right); //update wall
    array with new view;
286     visited[y][x] = 1;
287     if (x==4 && y==4)found_goal=1;
288     if(found_goal){
289         if(increment_v(x,y) && !maze_explore_complete){
290             maze_explore_complete=1;
291           //  printf("Explore Complete! Heading back to start...\n");
292         }
293         goal_x = v_x;
294         goal_y = v_y;
295     }
296     if(maze_explore_complete){
297         if(x==0 && y==0){
298          //   printf("back at start! Heading to finish...\n");
299             maze_return_start=1;
300         }else{
301             goal_x=0;
302             goal_y=0;
303         }
304         if(maze_return_start){
305             if(x==4 && y==4)return 4;
```

```
306             rolling_flag=1;
307             goal_x=4;
308             goal_y=4;
309         }
310 
311     }
312     initMaze();
313     floodfill(goal_x,goal_y,0);
314     cell_f_num = getFNum(x,y);
315     /*Ideally, dependent on orientation. Change it in later update*/
316     if(!isWallNorth(x,y)){
317         dir=0; //default to north
318         if((cell_f_num - getFNum(x,y+1))>=0)f_num_dif = cell_f_num - getFNum(x,y+1);
319     }
320     temp = cell_f_num - getFNum(x+1, y);
321     if(!isWallEast(x,y)&& (temp<f_num_dif) && temp>=0){ //if no wall east and the num is
    lower, go east
322         dir = 1;
323         f_num_dif= temp;
324     }
325     temp = cell_f_num - getFNum(x, y-1);
326     if(!isWallSouth(x,y) && temp<f_num_dif && temp>=0){ //if no wall east and the num is
    lower, go east
327         f_num_dif = temp;
328         dir = 2;
329     }
330 
331     temp = cell_f_num - getFNum(x-1, y);
332     if(!isWallWest(x,y)&& temp<f_num_dif && temp>=0){   //if no wall east and the num is
    lower, go east
333         f_num_dif = temp;
334         dir = 3;
335     }
336     if (dir>=0){
337         return dir;
338 
339     }else{
340         //error. nowhere to go...
341         return -1;
342     }
343 
344 }
345 
346 /* * * * *  * * * *  * * * ** * * * * * * * * * * * * * * *
347     map cell walls
348 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
349 int mapCell(int x, int y, int orientation, int front, int left, int right){ //0-N, 1-E, 2-
    S, 3-W
350     int l_wall=0;
351     int f_wall=0;
352     int r_wall=0;
353     /*Get Walls. If wall exists, set l_wall, f_wall, r_wall
354     *
355     ******************************************************/
356     //check if sensors are greater than wall present threshold
357     if(front>40){
358         f_wall = 1;
359     }
360 
361     if(left>45){
362         l_wall = 1;
363     }
```

```
364
365     if(right>45){
366         r_wall = 1;
367     }
368
369   //set north, east, south, west wall based off orientation
370     switch(orientation){
371         case 0:    //facing north
372             if (l_wall)setWall(x,y,3);
373             if (r_wall)setWall(x,y,1);
374             if (f_wall)setWall(x,y,0);
375             break;
376         case 1:
377             if (l_wall)setWall(x,y,0);
378             if (r_wall)setWall(x,y,2);
379             if (f_wall)setWall(x,y,1);
380             break;
381         case 2:
382             if (l_wall)setWall(x,y,1);
383             if (r_wall)setWall(x,y,3);
384             if (f_wall)setWall(x,y,2);
385             break;
386         case 3:
387             if (l_wall)setWall(x,y,2);
388             if (r_wall)setWall(x,y,0);
389             if (f_wall)setWall(x,y,3);
390             break;
391     }
392 }
393
394 int setWall(int x, int y, int dir){ //0-N, 1-E, 2-S, 3-W
395 /***********************************
396 *setWall
397 *
398 * globals: walls[][]
399 * inputs: x position to set wall in
400 *         y position to set wall in
401 *         direction to set wall
402 *
403 * output: valid wall array
404 *
405 * ***********************************/
406     int in_bounds = 0;
407     if(dir==0 && y<(MAZE_HEIGHT-1))in_bounds=1;
408     if(dir==1 && x<(MAZE_WIDTH-1))in_bounds=1;
409     if(dir==2 && y>0)in_bounds=1;
410     if(dir==3 && x>0)in_bounds=1;
411
412     if(in_bounds){
413         if (dir==0){        //NORTH
414             if(isWallEast(x,y))walls[y][x]=3;
415             else walls[y][x]=1;
416
417         }else if(dir==1){    //EAST
418             if(isWallNorth(x,y))walls[y][x]=3;
419             else walls[y][x]=2;
420         }else if(dir==2){    //SOUTH
421             if(isWallEast(x,y-1))walls[y-1][x]=3;
422             else walls[y-1][x]=1;
423
424         }else if(dir==3){    //WEST
425             if(isWallNorth(x-1,y))walls[y][x-1]=3;
```

```
426             else walls[y][x-1]=2;
427 
428         }
429     }
430 }
431 
432 int getFNum(int x, int y){
433 //get floodfill value
434     if(x<MAZE_WIDTH && y<MAZE_HEIGHT)return fnum[y][x];
435     else return -1;
436 }
437 
438 void setFNum(int x, int y, int num){
439 /*helper function to keep consistency with wall array*/
440     fnum[y][x]=num;
441 }
442 
443 
444 void floodfill(int x, int y, int level){
445 /***********************************************
446 *recursive function which outputs to FNum array the minimum
447 number of steps required to get to the goal from each cell. Takes
448 into account current state of wall array.
449 
450 when calling function, level should be 0
451 
452 inputs: goal x position
453     goal y position
454     current level (0 for first run)
455 returns: void
456 ************************************************/
457     setFNum(x,y,level);
458     ////printf("!isWallNorth")
459     //printf("!isWallNorth && ((level+1)< getFNum(x, y+1))-->%d", !isWallNorth &&
    ((level+1)< getFNum(x, y+1)));
460     if(!isWallNorth(x,y) && ((level+1)< getFNum(x, y+1))){
461         ////printf("Enter north flood\n");
462         floodfill(x,y+1,level+1);
463     }
464     if(!isWallEast(x,y) && ((level+1)< getFNum(x+1, y))){
465         ////printf("Enter east flood\n");
466         floodfill(x+1,y,level+1);
467     }
468     if(!isWallSouth(x,y) && ((level+1)< getFNum(x, y-1))){
469         ////printf("Enter south flood\n");
470         floodfill(x,y-1,level+1);
471     }
472     if(!isWallWest(x,y) && ((level+1)< getFNum(x-1, y))){
473         ////printf("Enter west flood\n");
474         floodfill(x-1,y,level+1);
475     }
476     return;
477 }
478 
479 /* * * * *  * * * *  * * * ** * * * * * * * * * * * * * * *
480 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
481 /*end floodfill.c*/
482 
483 void main(void){
484     SYNR = 0x02;
485     REFDV = 0x00;
486     while (!(CRGFLG & 0x08)){}
```

```
487     CLKSEL = 0x80;
488
489     InitializeATD();
490     //set up MDCU for 0.1ms = 0.0001 seconds interrupt
491     //SetupMDCU();
492     //set up MDCU for 0.1ms = 0.0001 seconds interrupt
493     MCCTL = 0xCF;  //interrupt enable, load latest mod reg, mcen, pre=16
494     MCCNT = 150;   //(0.0001-sec)(24000000-CLKperSEC)(16-CLKperCOUNT) = COUNT
495     MCCTL = 0xCF;
496     SetupDDR_Ports();
497     setup_buzzer();
498     //initMaze();
499     EnableInterrupts;
500     for(;;) {
501
502         ATD0CTL5 = 0x33;
503
504         while (!(ATD0STAT0 & 0x80));
505
506 /* * * * * * * * * * * * * * * * * * * *
507     Merging Drive and wall array and floodfill
508  * * * * * * * * * * * * * * * * * * * */
509     switch (state){
510         case 0:  G_Turn = explore(Current_x, Current_y, 4,4, G_Current_Orient, y_s,
    Output_left[OutputIndex], Output_right[OutputIndex]); // 4 finished, 0 -N, 1 -E, 2 -S, 3 -
    W, -1 -error
511                         SWITCH_DIRECTION();
512                         G_Steps = 390;
513                         RAMP_BOTH_UP();
514                         HALF_STEP_FORWARD(G_Steps);
515                         PTH = SEG7_DISPLAY[0];
516
517                         //if (rolling_flag) state = 6;
518                         //else
519                         state = 1;
520
521                         break;
522         case 1:
523                        // for rolling turn state = 6 or for stationary turn set new speed
524                         if (rolling_flag){
525                           speed_left = 22;       // int  26
526                           speed_right = 22;     // int  26
527                           top_speed = 22;      // 26
528                           //state = 6; break;
529                         }
530
531                         turn = 0;
532                         G_Turn = explore(Current_x, Current_y, 4,4, G_Current_Orient, y_s,
    Output_left[OutputIndex], Output_right[OutputIndex]); // 4 finished, 0 -N, 1 -E, 2 -S, 3 -
    W, -1 -error
533
534                         PTH = SEG7_DISPLAY[G_Turn];
535                         if (G_Turn == -1) break;
536                         else if (G_Turn == 4) state = 3;
537                         else if (G_Turn == G_Current_Orient){  // remain going straight
538                             SWITCH_DIRECTION();
539                             G_Steps = G_Block_Step;
540
541                             HALF_STEP_FORWARD(G_Steps);
542
543                         } else {  // turn or u-turn, ramp down, stationary turn, ramp up
544                             SWITCH_DIRECTION();
```

```
545                         state = 2;
546 
547                     }
548 
549                     break;
550 
551         case 2:   turn = 1;
552                     STATIONARY_TURN(G_Steps, direction);    // ramp down TURN - U-TURN
553 
554                     G_Steps = 360; // 360
555 
556                     RAMP_BOTH_UP();
557                     HALF_STEP_FORWARD(G_Steps);
558                     state = 1; //1
559                     break;
560 
561         case 3:  PTH = SEG7_DISPLAY[4];
562                     change_buzzer(0);
563                     break;
564 
565         case 4:
566 
567                     state = 1;
568                     //PTH = SEG7_DISPLAY[5];
569                     break;
570 
571         case 5 :  // for pivot turns
572                     PIVOT_TURN(350, direction);
573                     state = 1;
574                     //NEXT_STATE();
575                     break;
576 
577         case 6 :  // for rolling turns
578                     turn = 0;
579                     G_Turn = explore(Current_x, Current_y, 4,4, G_Current_Orient, y_s,
    Output_left[OutputIndex], Output_right[OutputIndex]); // 4 finished, 0 -N, 1 -E, 2 -S, 3 -
    W, -1 -error
580 
581                     PTH = SEG7_DISPLAY[6];
582                     if (G_Turn == -1) break;
583                     else if (G_Turn == 4) state = 3;
584                     else if (G_Turn == G_Current_Orient){  // remain going straight
585                         SWITCH_DIRECTION();
586                         G_Steps = G_Block_Step;
587 
588                         HALF_STEP_FORWARD(G_Steps);
589 
590                     } else {  // turn or u-turn, ramp down, stationary turn, ramp up
591                         SWITCH_DIRECTION();
592                        //if (G_Turn == G_90_Turn*2) state = 2;
593                        //else
594                        state = 7;
595                     }
596 
597                     break;
598 
599         case 7:  //ROLLING_TURN (set both speeds 1:3 ratio)
600                     ROLLING_TURN(580, direction);
601                     //G_Steps = 50;
602                     state = 6;
603                     break;
604 
```

```
605         }
606     }
607 } // end of main
608 
609 //interrupt triggers every 0.1ms
610 #pragma TRAP_PROC
611 #pragma CODE_SEG __SHORT_SEG NON_BANKED
612 interrupt VectorNumber_Vtimmdcu void mdcuInterrupt ()
613 {
614 
615     MCFLG |= 0x80;
616 
617     interruptcount_filter++;
618     interruptcount_left++;
619     interruptcount_right++;
620     //   fixed point math butterworth filter
621     //   read sensors every millisecond = 1ms = 0.001sec
622     if(interruptcount_filter == 10 ){ //&& ((state == 0) || (state == 1) || (state == 2))
623         interruptcount_filter = 0;
624         // right sensor
625         rx = ATD0DR2H;
626         rx <<= 14;
627         rdlng = rx - a[0] * rd[1] - a[1] * rd[2];
628         rdlng <<= 2;
629         rdlng >>= 16;
630         rd[0] = rdlng;
631         rylng = b[0] * rd[0] + b[1] * rd[1] + b[2] * rd[2];
632         rylng <<= 2;
633         rylng >>= 16;
634 
635         OutputIndex++;
636         if (OutputIndex == 2)
637             OutputIndex = 0;
638 
639         Output_right[OutputIndex] = rylng;
640         rd[2] = rd[1];
641         rd[1] = rd[0];
642 
643         // front sensor
644         x_s = ATD0DR1H;
645         x_s <<= 14;
646 
647         dlng = x_s - a[0] * d[1] - a[1] * d[2];
648         dlng <<= 2;
649         dlng >>= 16;
650         d[0] = dlng;
651         ylng = b[0] * d[0] + b[1] * d[1] + b[2] * d[2];
652         ylng <<= 2;
653         ylng >>= 16;
654         y_s = ylng;
655         d[2] = d[1];
656         d[1] = d[0];
657 
658         // left sensor
659         lx = ATD0DR0H;
660         lx <<= 14;
661 
662         ldlng = lx - a[0] * ld[1] - a[1] * ld[2];
663         ldlng <<= 2;
664         ldlng >>= 16;
665         ld[0] = ldlng;
666         lylng = b[0] * ld[0] + b[1] * ld[1] + b[2] * ld[2];
```

```
667         lylng <<= 2;
668         lylng >>= 16;
669 
670         Output_left[OutputIndex] = lylng;
671         ld[2] = ld[1];
672         ld[1] = ld[0];
673 
674     }//  end of sensor reading
675 
676 }//  end of MDCU
677 
678 void HALF_STEP_FORWARD(int steps){
679     while (count_both <= steps){
680         if(y_s>60){       // 55
681           count_both = 0;
682           return;
683         }
684         if (state == 1 || state == 0 || state == 2 || state==6) {
685             //PD CONTROL
686             diff = ((Output_left[OutputIndex] > Output_right[OutputIndex]) ? (
687             Output_left[OutputIndex] - Output_right[OutputIndex]) :
    (Output_right[OutputIndex] - Output_left[OutputIndex]));
688 
689             if ((Output_left[OutputIndex] < 50) && (Output_right[OutputIndex] < 50)){
690 
691                 // go straight
692                 speed_left = speed_right = top_speed;
693 
694                 // right speed time
695                 if ((interruptcount_right >= speed_right)){
696                   RIGHT_STEP_FORWARD();
697                   count_both++;
698                 }// end of right port writing
699 
700                 // left speed time
701                 if (interruptcount_left >= speed_left){
702                   LEFT_STEP_FORWARD();
703                 }// end of left port writing
704 
705             }
706             // left wall present correct right speed
707             else if ((Output_left[OutputIndex] > Output_right[OutputIndex]) && (diff > 10))
    {
708 
709                 //PTH = 0x79; // right one
710 
711                 error_r = 72.0 - Output_left[OutputIndex];
712                 errorDot_r = Output_left[OutputIndex] - Output_left[(OutputIndex - 1 == -1
    ? 1 : 0)];
713 
714                 speed_right = top_speed - (kp * error_r - kd * errorDot_r);
715 
716                 if (speed_right < top_speed - 7)
717                     speed_right = top_speed - 7;
718 
719                 // right speed time
720                 if ((interruptcount_right >= speed_right)){
721                   RIGHT_STEP_FORWARD();
722                   count_both++;
723                 }// end of right port writing
724 
725                 // left speed time
```

```
726                 if (interruptcount_left >= speed_left){
727                   LEFT_STEP_FORWARD();
728                   //count_both++;
729                 }// end of left port writing
730 
731                // right wall present correct left speed
732             } else if ((Output_right[OutputIndex] > Output_left[OutputIndex]) && (diff > 10)) {
733 
734                 //PTH = 0x4F; // left one
735                 error_l = 72.0 - Output_right[OutputIndex];
736                 errorDot_l = Output_right[OutputIndex] - Output_right[(OutputIndex - 1 == -1 ? 1 : 0)];
737 
738                 speed_left = top_speed - (kp * error_l - kd * errorDot_l);
739 
740                 if (speed_left < top_speed - 7)
741                     speed_left = top_speed - 7;
742 
743                 // right speed time
744                 if ((interruptcount_right >= speed_right)){
745                   RIGHT_STEP_FORWARD();
746                   //count_both++;
747                 }// end of right port writing
748 
749                 // left speed time
750                 if (interruptcount_left >= speed_left){
751                   LEFT_STEP_FORWARD();
752                   count_both++;
753                 }// end of left port writing
754 
755             } else if (diff <= 10) {   // go straight
756                 // go straight
757                 //PTH = 0x49; // both one
758                 speed_left = speed_right = top_speed;
759 
760                 // right speed time
761                 if ((interruptcount_right >= speed_right)){
762                   RIGHT_STEP_FORWARD();
763                   count_both++;
764                 }// end of right port writing
765 
766                 // left speed time
767                 if (interruptcount_left >= speed_left){
768                   LEFT_STEP_FORWARD();
769                 }// end of left port writing
770 
771             }// end go straight
772 
773         }// end of PD controller
774 
775     }// end of while
776     count_both = 0;
777       //state = 0;
778 
779 
780 
781 } // end of half step forward
782 
783 void SWITCH_DIRECTION(void){
784   switch (G_Current_Orient){
785     case 0:   G_Current_Orient = G_Turn;
```

```
786                   if (G_Turn == 0) Current_y++;  // straight
787                   else if (G_Turn == 1){ // right turn
788                     Current_x++;
789                     direction = 1;
790                     G_Steps = G_90_Turn;
791                   }
792                   else if (G_Turn == 2) {// u-turn
793                     Current_y--;
794                     G_Steps = G_90_Turn*2;
795                   }
796                   else if (G_Turn == 3){ // left turn
797                     Current_x--;
798                     direction = 0;
799                     G_Steps = G_90_Turn;
800                   }
801                 break;
802     case 1: { switch (G_Turn) { // East to
803                     case 0: G_Current_Orient=0; Current_y++; direction = 0; G_Steps =
    G_90_Turn; break; // North - left turn
804                     case 1: G_Current_Orient=1; Current_x++; break; // East - Straight
805                     case 2: G_Current_Orient=2; Current_y--; direction = 1; G_Steps =
    G_90_Turn; break; // South - right turn
806                     case 3: G_Current_Orient=3; Current_x--; G_Steps = G_90_Turn*2; break;
    // West - u-turn
807                     }
808     }break;
809
810     case 2:  { switch (G_Turn) { // South to
811                     case 0: G_Current_Orient=0; Current_y++; G_Steps = G_90_Turn*2; break;
    // North - u-turn
812                     case 1: G_Current_Orient=1; Current_x++; direction = 0; G_Steps =
    G_90_Turn; break; // East - left turn
813                     case 2: G_Current_Orient=2; Current_y--; break; // South - Straight
814                     case 3: G_Current_Orient=3; Current_x--; direction = 1; G_Steps =
    G_90_Turn; break; // South - right turn
815                     }
816     }break;
817
818     case 3:  { switch (G_Turn) { // West to
819                     case 0: G_Current_Orient=0; Current_y++; direction = 1; G_Steps =
    G_90_Turn; break; // North - right turn
820                     case 1: G_Current_Orient=1; Current_x++; G_Steps = G_90_Turn*2; break;
    // East - u-turn
821                     case 2: G_Current_Orient=2; Current_y--; direction = 0; G_Steps =
    G_90_Turn; break; // South - left turn
822                     case 3: G_Current_Orient=3; Current_x--; break; // South
823                   }
824     }break;
825   }
826 }
827 void RIGHT_STEP_FORWARD(void){
828
829     //right wheel speed
830         PORTB =  (PORTB & ~0xF0) |  (RightStepArray[stepindex_right] & 0xF0);
831         stepindex_right++;
832
833         // reset step index
834         if (stepindex_right > 7)
835             stepindex_right = 0;
836
837         // reset right wheel interrupt count
838         interruptcount_right = 0;
```

```
839 }
840 void LEFT_STEP_FORWARD(void){
841 
842     // left speed time
843         PORTB =  (PORTB & ~0x0F) |  (LeftStepArray[stepindex_left] & 0x0F);
844         stepindex_left++;
845 
846         // reset step index
847         if (stepindex_left > 7)
848             stepindex_left = 0;
849 
850         //reset left wheel interrupt count
851         interruptcount_left = 0;
852 }
853 
854 void STATIONARY_TURN(int steps, int direction){
855 
856     // slow down robot to a stop
857     RAMP_BOTH_DOWN();
858 
859     // rotate in place
860     RAMP_BOTH_UP_REV(steps/4, direction);
861     HALF_STEP_REV(steps/2, direction);
862     RAMP_BOTH_DOWN_REV(steps/4, direction);
863 
864     // accelerate to top speed
865     // RAMP_BOTH_UP();
866 
867 } // end of u-turn
868 
869 void RAMP_BOTH_UP_REV(int steps, int direction){
870 
871     while (count_both <= steps){
872 
873         if (current_speed >= top_speed) {
874             if ((interruptcount_right >= current_speed) && (interruptcount_left >=
    current_speed)){
875 
876                 PORTB =  (LeftStepArray[stepindex_left] | RightStepArray[stepindex_right]);
877 
878                 if (direction){    // if direction is right
879                     stepindex_right--;
880                     stepindex_left++;
881                 } else {        // if direction is left
882                     stepindex_right++;
883                     stepindex_left--;
884                 }
885 
886                 // reset step index
887                 if (stepindex_right > 7)
888                     stepindex_right = 0;
889                 if (stepindex_right < 0)
890                     stepindex_right = 7;
891                 // reset step index
892                 if (stepindex_left > 7)
893                     stepindex_left = 0;
894                 if (stepindex_left < 0)
895                     stepindex_left = 7;
896 
897                 // reset left wheel interrupt count
898                 interruptcount_left = 0;
899                 // reset right wheel interrupt count
```

```
900                 interruptcount_right = 0;
901 
902                 if (current_speed > top_speed)
903                     current_speed--;
904 
905                 count_both++;
906             }
907         }
908 
909     }    // end while
910 
911     current_speed = top_speed;
912     count_both = 0;
913 
914 }// end of RAMP_BOTH_UP_REV
915 
916 void HALF_STEP_REV(int steps, int direction){    // used for stationary turning
917 
918     while (count_both <= steps){
919         if ((interruptcount_right >= top_speed) && (interruptcount_left >= top_speed)){
920 
921             PORTB = (RightStepArray[stepindex_right] | LeftStepArray[stepindex_left]);
922 
923             if (direction){ // direction is right
924                 stepindex_right--;
925                 stepindex_left++;
926             } else {
927                 stepindex_right++;
928                 stepindex_left--;
929             }
930 
931             // reset step index
932             if (stepindex_right > 7)
933                 stepindex_right = 0;
934             if (stepindex_right < 0)
935                 stepindex_right = 7;
936             // reset step index
937             if (stepindex_left > 7)
938                 stepindex_left = 0;
939             if (stepindex_left < 0)
940                 stepindex_left = 7;
941 
942             //reset left wheel interrupt count
943             interruptcount_left = 0;
944             // reset right wheel interrupt count
945             interruptcount_right = 0;
946 
947             count_both++;
948         }
949     }
950     count_both = 0;
951 }// end of HALF_STEP_REV
952 
953 void RAMP_BOTH_DOWN_REV(int steps, int direction){
954 
955     while (count_both <= steps){
956 
957         if (current_speed <= bottom_speed) {
958             if ((interruptcount_right >= current_speed) && (interruptcount_left >=
    current_speed)){
959 
960                 PORTB = (LeftStepArray[stepindex_left] | RightStepArray[stepindex_right]);
```

```
961
962                 if (direction){    // if direction is right
963                     stepindex_right--;
964                     stepindex_left++;
965                 } else {        // if direction is left
966                     stepindex_right++;
967                     stepindex_left--;
968                 }
969
970                 // reset step index
971                 if (stepindex_right > 7)
972                     stepindex_right = 0;
973                 if (stepindex_right < 0)
974                   stepindex_right = 7;
975                 // reset step index
976                 if (stepindex_left > 7)
977                     stepindex_left = 0;
978                 if (stepindex_left < 0)
979                     stepindex_left = 7;
980
981                 // reset left wheel interrupt count
982                 interruptcount_left = 0;
983                 // reset right wheel interrupt count
984                 interruptcount_right = 0;
985
986                 if (current_speed < bottom_speed)
987                     current_speed++;
988
989                 count_both++;
990             }
991         }
992
993     }// end of RAMP_BOTH_DOWN_REV
994
995         current_speed = bottom_speed;
996         count_both = 0;
997
998 } // end of
999
1000 void RAMP_BOTH_UP(void){
1001
1002     //current_speed = bottom_speed;
1003     while (current_speed >= top_speed){
1004
1005         if ((interruptcount_right >= current_speed) && (interruptcount_left >=
     current_speed)){
1006
1007             // this works for moving left & right wheel at same rate
1008             PORTB =  (LeftStepArray[stepindex_left] | RightStepArray[stepindex_right]);
1009
1010             stepindex_right++;
1011             stepindex_left++;
1012
1013             // reset step index
1014             if (stepindex_right > 7)
1015                 stepindex_right = 0;
1016             // reset step index
1017             if (stepindex_left > 7)
1018                 stepindex_left = 0;
1019
1020             //reset left wheel interrupt count
1021             interruptcount_left = 0;
```

```
1022            // reset right wheel interrupt count
1023            interruptcount_right = 0;
1024
1025            current_speed--;
1026        }
1027
1028    }
1029
1030    current_speed = top_speed;
1031
1032
1033 }    // end RAMP_BOTH_UP
1034
1035 void RAMP_BOTH_DOWN(void){
1036
1037    while (current_speed <= bottom_speed){
1038
1039        if ((interruptcount_right >= current_speed) && (interruptcount_left >=
     current_speed)){
1040
1041            //PTH = 0x24;   //2
1042            PORTB =  (LeftStepArray[stepindex_left] | RightStepArray[stepindex_right]);
1043
1044            stepindex_right++;
1045            stepindex_left++;
1046
1047            // reset step index
1048            if (stepindex_right > 7)
1049                stepindex_right = 0;
1050            // reset step index
1051            if (stepindex_left > 7)
1052                stepindex_left = 0;
1053
1054            //reset left wheel interrupt count
1055            interruptcount_left = 0;
1056            // reset right wheel interrupt count
1057            interruptcount_right = 0;
1058
1059            current_speed++;
1060        }
1061
1062    }
1063
1064    current_speed = bottom_speed;
1065    //state = 0;
1066
1067 }
1068 /* */
1069 void RAMP_ONE_DOWN(int steps, int direction){    // might want to control the amount of
     steps it takes
1070
1071   while (count_one <= steps){
1072
1073
1074        if (direction){    // if direction = 1 (turning right)
1075          if (current_speed < bottom_speed){
1076            if (interruptcount_right >= current_speed){    // slow down right wheel
     (turning right)
1077
1078                RIGHT_STEP_FORWARD();
1079                current_speed = current_speed + 2;
1080
```

```
1081            }
1082          }
1083            if (interruptcount_left >= top_speed){    // constant speed
1084
1085                LEFT_STEP_FORWARD();
1086                count_one++;
1087            }
1088        } else {    // if direction is false
1089            if (current_speed < bottom_speed){
1090            if (interruptcount_left >= current_speed){    // slow down left wheel (turning
    left)
1091
1092                LEFT_STEP_FORWARD();
1093                current_speed = current_speed + 2;
1094
1095            }
1096          }
1097            if (interruptcount_right >= top_speed){    // constant speed
1098
1099                RIGHT_STEP_FORWARD();
1100                count_one++;
1101            }
1102        }
1103   }
1104
1105     count_one = 0;
1106     current_speed = bottom_speed;
1107
1108 }
1109
1110 void RAMP_ONE_UP(int steps, int direction){    // might want to control the amount of steps
    it takes
1111
1112     while (count_one <= steps){
1113
1114         if (direction){    // if direction = 1 (turning right)
1115            if (current_speed > top_speed){
1116              if (interruptcount_right >= current_speed){    // accelerate right wheel
    (turning right)
1117
1118         RIGHT_STEP_FORWARD();
1119                 current_speed = current_speed - 2;
1120
1121            }
1122          } else {
1123            RIGHT_STEP_FORWARD();
1124          }
1125            if (interruptcount_left >= top_speed){    // constant speed
1126
1127                LEFT_STEP_FORWARD();
1128                count_one++;
1129            }
1130        } else {    // if direction is false
1131            if (current_speed > top_speed){
1132            if (interruptcount_left >= current_speed){    // accelerate left wheel (turning
    left)
1133
1134                LEFT_STEP_FORWARD();
1135                current_speed = current_speed - 2;
1136
1137            }
1138          } else {
```

```
1139            LEFT_STEP_FORWARD();
1140          }
1141            if (interruptcount_right >= top_speed){    // constant speed
1142
1143                RIGHT_STEP_FORWARD();
1144                count_one++;
1145            }
1146        }
1147   }
1148
1149     count_one = 0;
1150     current_speed = top_speed;
1151 }
1152
1153 void PIVOT_TURN(int steps, int direction){
1154
1155     // slow down one wheel
1156     RAMP_ONE_DOWN(steps, direction);
1157
1158     // rev up from stop the one wheel
1159     RAMP_ONE_UP(steps/2, direction);
1160
1161 }
1162   /* */
1163 void ROLLING_TURN(int steps, int direction){
1164
1165 count_one = 0;
1166 while (count_one <= steps) { // change steps to discrete value
1167
1168     // set speeds
1169     speed_left = (direction) ? top_speed : (3*top_speed);
1170     speed_right = (direction) ? (3*top_speed) : top_speed;
1171
1172     if (direction){
1173
1174       if (interruptcount_right >= speed_right)
1175         RIGHT_STEP_FORWARD();
1176       if (interruptcount_left >= speed_left){
1177         LEFT_STEP_FORWARD();
1178         count_one++;
1179       }
1180     } else {
1181       if (interruptcount_right >= speed_right){
1182         RIGHT_STEP_FORWARD();
1183         count_one++;
1184       }
1185       if (interruptcount_left >= speed_left)
1186         LEFT_STEP_FORWARD();
1187     }
1188 }
1189   count_one = 0;
1190     speed_left = speed_right = top_speed;
1191
1192 }
1193
1194 void NEXT_STATE(void){
1195
1196   switch (state){
1197       case 0 : state = 1; break;
1198       case 1 : state = 2; break;
1199       case 2 : state = 3; break;
1200       case 3 : state = 4; break;
```

```
1201       case 4 : state = 5; break;
1202       case 5 : state = 6; break;
1203       case 6 : state = 7; break;
1204     case 7 : state = 8; break;
1205       case 8 : state = 1; break;
1206 
1207   }
1208 
1209 }
1210 
1211 void InitializeATD(void)
1212 {
1213     ATD0CTL2 = 0x80; // power on ATD
1214     ATD0CTL3 = 0x18; // two conversions = 0x10 three conversions = 0x18
1215     ATD0CTL4 = 0x85; // was 0x84
1216 }
1217 
1218 void SetupMDCU(void){
1219     //set up MDCU for 0.1ms = 0.0001 seconds interrupt
1220     MCCTL = 0xCF;  //interrupt enable, load latest mod reg, mcen, pre=16
1221     MCCNT = 150;   //(0.0001-sec)(24000000-CLKperSEC)(16-CLKperCOUNT) = COUNT
1222     MCCTL = 0xCF;
1223 }
1224 
1225 void SetupRTI(void){
1226 
1227     RTICTL = 0b01101011;    //sets prescaler of 82^10 resulting in interrupt every 49.15ms
1228     CRGINT = 0x80;            //tells that want interrupt given, not flag raised
1229 
1230 }
1231 
1232 void SetupDDR_Ports(void){
1233 
1234     // set up 7-segment display
1235     DDRH = 0xFF;
1236     // set up motor ports
1237     DDRB = 0xFF; //     PORTB0-PORTB3  as output
1238     DDRP = 0x0F; //     PORTP0 and PORTP1 as output for 12EN=1 and 34EN=1
1239 
1240     PORTB=0b00000000;//  start with all off
1241     PTP=0b00001111; //  Turn on both 12EN and 34EN Enables for 754410 chip
1242 
1243 }
1244 
1245 void setup_buzzer(){
1246 
1247   PWMPRCLK=0x04; //ClockA=Fbus/2**4=24MHz/16=1.5MHz
1248     PWMSCLA=10;      //ClockSA=1.5MHz/2x125=6000 Hz
1249     PWMCLK=0b00100000;      //ClockSA for chan 5
1250     PWMPOL=0x20;              //high then low for polarity
1251     PWMCAE=0x0;              //left aligned
1252     PWMCTL=0x0;             //8-bit chan, pwm during freeze and wait
1253     PWMPER5=100;      //PWM_Freq=ClockSA/100=6000Hz/100=60Hz. CHANGE THIS
1254     PWMDTY5=50;      //50% duty cycle                        AND THIS TO GET DIFFERENT
     SOUND
1255     PWMCNT5=0;         //clear initial counter. This is optional
1256     //PWME=0x20;        //Enable chan 5 PWM
1257 
1258 }
1259 
1260 void change_buzzer(int dout){
1261 
```

```
1262      if(dout == 0){
1263        PWMSCLA=10; //10
1264        PWME=0x20;        //Enable chan 5 PWM
1265      }else if(dout == 1){
1266        PWMSCLA=30; // 30
1267        PWME=0x20;        //Enable chan 5 PWM
1268      }else if(dout == 2){
1269        PWMSCLA=50; // 50
1270        PWME=0x20;        //Enable chan 5 PWM
1271      }else if(dout == 3){
1272        PWMSCLA=70; //70
1273        PWME=0x20;        //Enable chan 5 PWM
1274      }else if(dout == 4){
1275        PWMSCLA=90; //90
1276        PWME=0x20;        //Enable chan 5 PWM
1277      }
1278
1279 }
```