**Proyecto Final.**

**Objetivo:**

Implementar un robot que resuelva un laberinto usando el algoritmo flood fill y a su vez enviar datos al computador usando la interfaz serial para ser graficados en la plataforma Processing.

**Objetivos Específicos:**

* Acondicionar la tarjeta de desarrollo Arduino MEGA 2560 al vehículo de forma que pueda ejercer la ejecución o intercambio de señales con el mismo, de acuerdo a valores censados con la tarjeta QTR8C,ejecutar acciones, que se vean traducidas de acuerdo al software implementado en movimientos lógicos que contribuyan a la resolución del laberinto propuesto.
* Diseñar e implementar un laberinto físico con paredes y caminos con líneas negras, este con 4 entradas diferentes que dirige a un mismo centro.

**Acción del código:**

Se define una cuadricula con un tamaño de 6x6 debido a las dimensiones del laberinto, creando las respectivas paredes (con la función setwall) con determinadas funciones de acuerdo a la matriz, también se definen los valores asignados que irán siendo actualizados para cada posición, reconociéndola para mandar números donde se podrán ver en la cuadricula para así, con estos, direccionar el robot con números según la posición y reasignando valores donde este encuentre una pared.

***Explicación del algoritmo Flood fill***

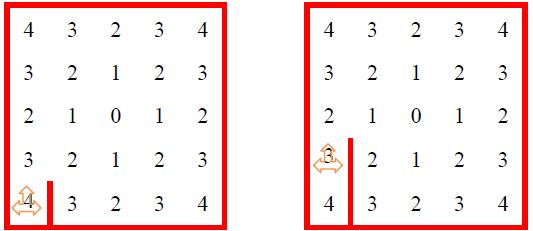
Es más fácil identificarlo como el algoritmo de inundación, ya que se asignan valores numéricos a las celdas del laberinto para con ello tener un estimado de cuando lejos se encuentra el robot de la salida del laberinto y con ello determinar la ruta que lo dirija a ella directamente, el algoritmo se basa en dividir el laberinto en celdas, es decir, como una matriz con diferentes casillas, y los valores que estarán en cada casilla se van a calcular a partir de la casilla considerada como solución.

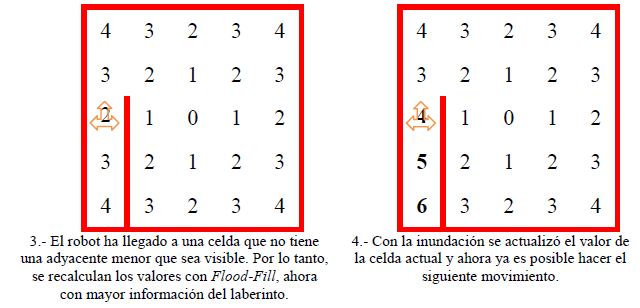
La solución o celda determinada como llegada tendrá el mínimo valor posible (valor 0) , es decir 0 pasos necesarios para alcanzar la solución, y las celdas adyacentes a estas, tanto de manera vertical como horizontal se les asignara el valor de 1 siendo entonces este el primer nivel, las celdas adyacentes a estas serán asignadas con el número 2, correspondiendo entonces al nivel 2 y así sucesivamente.

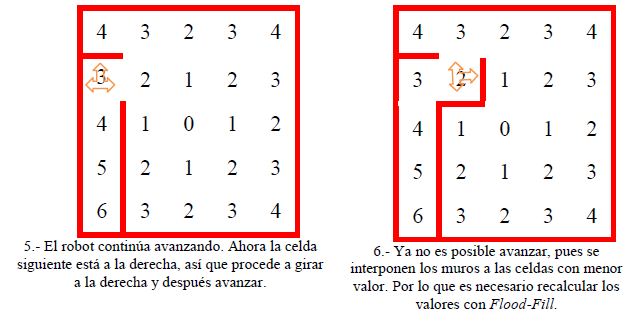
La información inicial proporcionada con la que parte el robot corresponde a las dimensiones que tiene el laberinto a recorrer y la posición (x , y) de la celda objetivo.

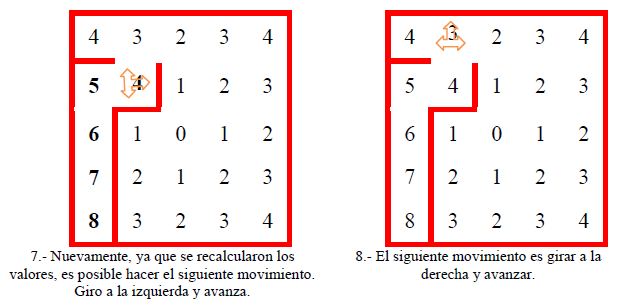
Inicialmente el robot no sabe el camino hacia la meta final por lo que deberá recorrerlo mediante la utilización de los sensores y motores con los que está equipado, el avance será dado paso a paso , es decir, celda a celda, con movimientos verticales y horizontales.

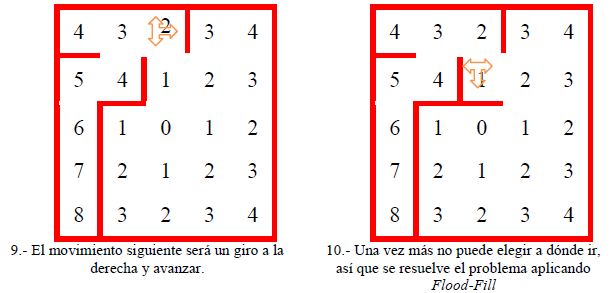
Para hacerlo más grafico la forma en que se moverá el robot dentro de laberinto corresponde a un patrón como el mostrado en la siguiente secuencia

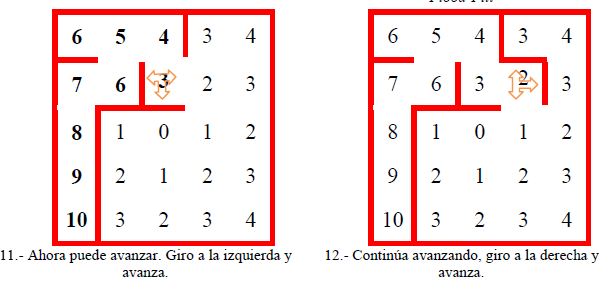
.

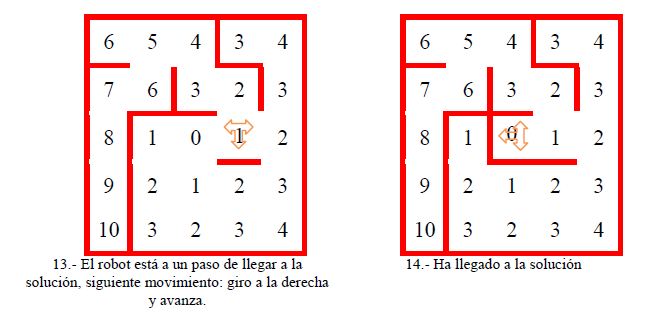












El siguiente corresponde a las líneas de código que se están empleando para la ejecución del algoritmo Flood fill, aunque se debe aún perfeccionar un poco.

1. #define BOARD\_WIDTH  6
2. #define BOARD\_HEIGHT 6
4. **struct** MAP
5. {
6. unsigned **char** b[BOARD\_HEIGHT][BOARD\_WIDTH];
7. } MAP;
8. **void** flood\_loop(struct MAP \*map, **int** x, **int** y,
9. unsigned **int** dst\_c, unsigned  **int** src\_c)
10. {
11. **int** fillL, fillR, i;
12. **int** in\_line = 1;


16. fillL = fillR = x;
17. **while**( in\_line )
18. {
19. map->b[y][fillL] = dst\_c;
20. fillL--;
21. in\_line = (fillL < 0) ? 0 : (map->b[y][fillL] == src\_c);
22. }
23. fillL++;
25. in\_line = 1;
26. **while**( in\_line )
27. {
28. map->b[y][fillR] = dst\_c;
29. fillR++;
30. in\_line = (fillR > BOARD\_WIDTH-1) ? 0 : (map->b[y][fillR] == src\_c);
31. }
32. fillR--;
34. **for**(i = fillL; i <= fillR; i++)
35. {
36. **if**( y > 0 && map->b[y - 1][i] == src\_c )
37. flood\_loop(map, i, y - 1, dst\_c, src\_c);
38. **if**( y < BOARD\_HEIGHT-1 && map->b[y + 1][i] == src\_c )
39. flood\_loop(map, i, y + 1, dst\_c, src\_c);
40. }
41. }
42. **void** flood\_fill(struct MAP \*map, **int** x, **int** y, unsigned **int** c)
43. {
44. flood\_loop(map, x, y, c, map->b[y][x]);
45. map->b[y][x] = c;
46. }

Y para el movimiento del robot se están empleando líneas como las usadas en en proyecto numero 2 , por ejemplo la siguiente función se emplea para definir los puntos de giro

void turn(char dir)

{

switch(dir)

{

case 'L':

digitalWrite(dir\_a, HIGH);

analogWrite(pwm\_a, turnSpeed);

digitalWrite(dir\_b, LOW);

analogWrite(pwm\_b, turnSpeed);

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[6] <200) // Espere a que sensor mas externo encuentre la línea.

{

line\_position = qtrrc.readLine(sensorValues);

}

// slow down speed

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// find center

while (line\_position > 4350) // esperar a que la posición\_de\_línea de encontrar cerca del centro.

{

line\_position = qtrrc.readLine(sensorValues);

}

// stop both motors

analogWrite(pwm\_b, 0); analogWrite(pwm\_a, 0);

break;

case 'R':

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, turnSpeed);

digitalWrite(dir\_b, HIGH);

analogWrite(pwm\_b, turnSpeed);

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[1] <200)

{

line\_position = qtrrc.readLine(sensorValues);

}

// slow down speed

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// find center

while (line\_position < 3250)

{

line\_position = qtrrc.readLine(sensorValues);

}

// stop both motors

analogWrite(pwm\_a, 0);

analogWrite(pwm\_b, 0);

break;

case 'B':

digitalWrite(dir\_a, LOW);

analogWrite(pwm\_a, turnSpeed);

digitalWrite(dir\_b, HIGH);

analogWrite(pwm\_b, turnSpeed);

line\_position = qtrrc.readLine(sensorValues);

while (sensorValues[1] <200) {

line\_position = qtrrc.readLine(sensorValues);

}

analogWrite(pwm\_a, turnSpeedSlow);

analogWrite(pwm\_b, turnSpeedSlow);

// find center

while (line\_position < 3250) {

line\_position = qtrrc.readLine(sensorValues);

}

// stop both motors

analogWrite(pwm\_a, 0);

analogWrite(pwm\_b, 0);

break;

case 'S':

// do nothing

break;

}

De igual forma se esta empleando la siguiente función

char select\_turn(unsigned char found\_left, unsigned char found\_straight, unsigned char found\_right)

{

// Make a decision about how to turn. The following code

// implements a left-hand-on-the-wall strategy, where we always

// turn as far to the left as possible.

//Tomar una decisión acerca de cómo hacer. El siguiente código implementa una estrategia de la izquierdo-mano-en-la-pared,

//

if(found\_left)

return 'L';

else if(found\_straight)

return 'S';

else if(found\_right)

return 'R';

else

return 'B';

}

Donde estas letras corresponden a los giros que se realizaran, por ejemplo L es un giro a la izquierda, R, un giro a la derecha, S seguir derecho, y B un giro de 180°.