Capítulo 5 Diseño e implementación de algoritmos

Los algoritmos diseñados se escribirán en forma de pseudocódigo, para cada algoritmo se muestran fragmentos de código representativo en el lenguaje de desarrollo NQC. Los algoritmos descritos a continuación están orientados a resolver los requerimientos del capítulo 4.

5.1 Diseño del algoritmo de representación de la receta

La receta una vez leída, está conformada por 5 variables, donde cada una representa un valor. Se realizó la consideración que si se deseara en un futuro extender la solución a la toma de todos los cubos en el orden proporcionado, la representación lógica de la receta debe de estar orden.

La representación lógica de la receta es un arreglo de 5 posiciones, donde un 0 representa un ingrediente de color negro y 1, un ingrediente de color azul; los valores son guardados en el orden de lectura, es así como el primer ingrediente leído queda en la posición 0 del arreglo con un valor 0 ó 1, y así sucesivamente para todos los ingredientes.

La comunicación en los robots es inestable, ya que pueden generarse interferencias cuando se transmite un mensaje. Uno de los factores por los cuales se pierde un mensaje es que ambos robots envían un mensaje al mismo tiempo [Overmars, 2002]. Dado el riesgo que representa enviar un mensaje, los cinco ingredientes de la receta deben ser codificados en un solo mensaje que se transmita al otro robot.

El algoritmo planteado para solucionar el problema, se muestra a continuación:

1. Leer la receta y guardar los 5 valores en un arreglo.
2. Tratar al arreglo como si fuese un número binario.

3. Declarar un arreglo de 5 posiciones y llenarlo de la siguiente manera:
    a. values[0]=16
    b. values[1]=8
    c. values[2]=4
    d. values[3]=2
    e. values[4]=1
4. Recorrer el arreglo de la receta a partir del índice i = 0 hasta el tamaño del arreglo; para cada valor en i multiplicarlo por el valor en i del arreglo de valores decimales e ir sumando estos resultados en una variable.
5. Si el número resultante es 0, convertirlo a 32.

Los mensajes que se envían a otro RCX son de tipo numérico y se encuentran en el rango de 0-255. El 0 es el valor que se asigna por defecto al *buffer* del robot, es decir que cuando se trata de leer un mensaje, devuelve 0 si no ha recibido ninguno; de ahí la necesidad de transformar el 0 a 32, donde la representación binaria del 32 es 100000, el cual es un valor que nunca se alcanzará ya que sólo se tienen 5 ingredientes.

En el programa son dos funciones las que implementan el algoritmo descrito, y se presentan a continuación.

```
void assign_values()
{
        values[0]=16;
         values[Cruz,2005]=8;
         values[UFMA,2005]=4;
         values[Savage,2005]=2;
         values[IEEE,2005]=1;
}

void decimalValue()
{
         for(index=0;index<SIZE;index++)
         {
                 send_ingredient_man=send_ingredient_man+(values[index]*colors_MAN[index]);
         }
```

```
        if(send_ingredient_man==0)
        send_inredient_man=32;
}
```

En el lado del RCX del refrigerador se ejecuta el proceso inverso, convirtiendo el valor decimal a un número binario. El programa en el RCX del robot M se muestra a continuación:

```
//Inicia el arreglo en 0 en caso de llegar recetas
//con ceros a la izquierda
void init_Array()
{
        for(index=0;index<SIZE;index++)
        {
                colors_REFRI[index]=0;
        }
}

//Convierte del numero decimal recibido a un binario
//que representa la receta y cuanta los cubos que se
//requieren
void start_convert_REFRI()
{
        init_Array();
        if(recive_message==32)
        {
                n_blue=0;
                n_black=SIZE;
        }
        else
        {
                index=SIZE-1;
                int one=0;
                int binary=0;
                while(recive_message!=0)
               {
                        binary=recive_message%2;
                        colors_REFRI[index]=binary;
                        recive_message=recive_message/2;
                        if(binary==1)
                        {
                                one++;
                        }
                        index--;
               }
               n_blue=one;
               n_black=SIZE-one;
        }
}
```

5.2 Diseño del algoritmo de comunicación

La comunicación se manejará en dos vertientes: una de ellas asegurará que el mensaje sea recibido y otra enviará mensajes rápidos; se usa cada algoritmo dependiendo de la situación. Si se desea que el mensaje llegue en menor tiempo, usamos un algoritmo de comunicación rápido; si se desea que el valor del mensaje llegue correctamente y se dé una confirmación, se usa la segunda vertiente.

En el apartado anterior se mencionó la inestabilidad del proceso de comunicación entre dos robots, por este motivo se plantea un algoritmo de comunicación que permita verificar si se ha recibido el mensaje. Esto se logra con ayuda de un mensaje de confirmación o *acknowledge*; el diagrama de este tipo de comunicación se plantea en la figura 5.1 donde M es el robot M y RI el RCX del robot R encargado de la comunicación, de la receta.

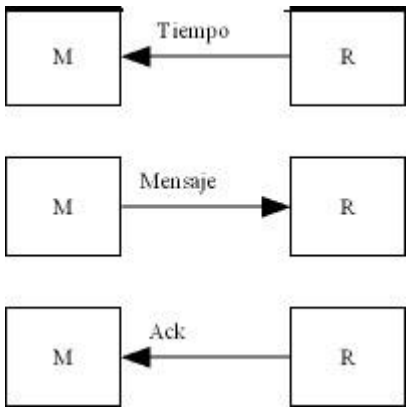

Figura 5.1 Diagrama de comunicación entre M y RI

Para evitar que ambos robots envíen mensajes al mismo tiempo, se utilizó el concepto de sincronización, implementándolo con ayuda de los *Timers* internos del RCX que son relojes que llevan el tiempo de un programa ejecutándose, estos es posible reiniciarlos en 0 cada vez que se desee. En el apéndice C se muestran los dos tipos de comunicación en dos diagramas de interacción.

El valor de 250 milisegundos se basa en pruebas empíricas realizadas para determinar el tiempo necesario para transmitir un valor. La prueba consistió en colocar a dos RCX en distintas posiciones a una distancia de 10 cm. y medir el tiempo que tomaba recibir el mensaje transmitido. Se tomo el valor más alto y se le sumaron 10 milisegundos como valor de tolerancia. Es importante aclarar que posiciones tales como los dos RCX dándose la espalda el mensaje nunca fue recibido.

El algoritmo de comunicación esta planteado de forma atómica, es decir, si se desea enviar más de un mensaje se debe ejecutar la rutina tantos mensajes se quieran enviar.

El algoritmo de comunicación queda expresado en pseudocódigo de la siguiente manera, resaltando el hecho que este algoritmo se ejecuta parte en el RCX del robot M y parte en el RCX del robot R:

1. R intenta comunicarse con M enviando el tiempo que lleva transmitiendo el mensaje por un periodo de 250 milisegundos.
2. R deja de enviar de enviar el mensaje durante 250 milisegundos escuchando constantemente.
3. Si no se ha recibido ningún mensaje, se reinicia el *Timer* en 0 y se ejecuta nuevamente el primer paso.
4. M recibe el tiempo que lleva ejecutándose el paso 1 en M, y espera a que se cumplan los 250 milisegundos que es el tiempo que M está en escucha.
5. M envía el valor decimal de la receta durante 250 milisegundos.
6. R recibe el mensaje y espera 250 milisegundos.
7. R envía un mensaje de que ha recibido la receta.
8. M escucha durante 250 milisegundos, esperando la confirmación.

El fragmento de código encargado de la comunicación en el RCX del robot M queda como sigue:

```
//Este método es el encargado de sincronizar los tiempos
// de los robots para que exista una comunicación sin
//interrupciones
//En la secuencia global de comunicación este es el paso
//2,6
void message_recive(int &target)
{
	ClearMessage();
	while(true)
	{
		if((target<Message()) && (Message()<(INTERVAL+target)))
		{
			Wait(((INTERVAL-(Message()-target))*10));
			break;
		}
	}
}

//Envía un mensaje a otro robot
//En la secuencia global de comunicación este es el paso
//3
void message_send(int &message)
{
	ClearTimer(0);
	ClearMessage();
	while(Timer(0)<=INTERVAL)
	{
		SendMessage(message);
		Wait(10);
	}
}

//Este método invoca a los métodos necesarios para
//la comunicación del robot M con otro
//Los parámetros son el target a que robot se dirige
//y cual es el mensaje
void start_com_receive(int &target,int message)
{
	message_recive(target);
	message_send(message);
	message_recive(target);
}

//Inicia la secuencia de comunicación hacia el robot RI, en el envió de la
//receta

void reciveMessage_from_REFRI_I()
{
	target=0;
	start_com_receive(target,send_ingredient_man);
}
```

Se maneja un intervalo en caso de que más adelante se desee ampliar la comunicación y se quiere diferenciar entre un tipo de mensaje y otro, en este caso el intervalo de mensajes va de 0-25, es por eso que la variable *target* vale 0, si se desea utilizar otro intervalo en la variable *target*, se coloca el recorrido que se debe de hacer, es decir, si se desea que ahora el mensaje de sincronización se envié en un rango de mensajes de 10-35, la variable *target* valdría 10. El número que se indica como secuencia como parte del comentario del método en el código, es en el orden en que se deben de llamar los métodos desde el gestor de comunicación. Es una secuencia global para ambos gestores de los robots, es decir, el paso siguiente puede encontrarse en el programa del otro RCX.

Se utilizan 25 *Timers* ya que después de haber realizado distintas pruebas, se concluyó que es el tiempo máximo que tarda un robot en recibir un mensaje; después de ese intervalo es poco probable que el otro robot reciba el mensaje.

Para el robot R, los métodos encargados de la comunicación quedan codificados de la siguiente forma:

```
//Este método se ejecuta primero para ver si encuentra
//alguna respuesta, envía su tiempo para sincronizar
//durante 250 ms que se define en la constante INTERVAL
//En la secuencia global de comunicación este es el paso
//1,5
void send_time(int &target)
{
        ClearTimer(0);
        ClearMessage();
        while(Timer(0)+target<=INTERVAL+target)
        {
                SendMessage(Timer(0)+target);
                Wait(10);
        }
}

//Este método se ejecuta después de send_time
//sirve para escuchar si existe alguna respuesta por
//parte del otro robot
//En la secuencia global de comunicación este es el paso
```

```
//4
void first_recive(int &target)
{
	ClearMessage();
	ClearTimer(0);
	while(Timer(0)+target<=INTERVAL+target)
	{
		if(Message()!=0)
		{
			recive_message=Message();
		}
		Wait(10);
	}
}
```

La otra vertiente de la comunicación es la comunicación rápida; este tipo de comunicación es la usada entre los dos RCX que se encuentran en el robot R. Los dos RCX se encuentran a una distancia más corta de la que se encuentran los dos robots cuando se quiere transmitir la receta; aunado a esto, los mensaje deben de llegar rápidamente porque se trata de indicadores de activación que deben de ser inmediatos de un robot a otro, cuando se tiene cierto tipo de lectura de los sensores. Es por ello que se diseña un algoritmo sin mensaje de confirmación pero evitando que ambos robots se encuentren transmitiendo al mismo tiempo.

Para sacar el mayor provecho del algoritmo y evitar tiempos muertos, se usa el concepto de multitareas, representado por distintos *tasks* en NQC que son lanzados con la sentencia *start*.

El pseudocódigo presentado a continuación es ejecutado más de una vez por ambos robots; en cada caso los RCX RI y RII juegan un papel distinto, siendo emisores o receptores de mensajes.

1. El receptor ejecuta una tarea que consiste en estar siempre en escucha de un nuevo mensaje.

2. El emisor envía un mensaje durante 250 milisegundos, el valor del mensaje es 120 que indica que debe de cambiar de actividad.
3. El receptor recibe el mensaje y termina la actividad que se esté ejecutando, para pasar a otro proceso.

El código presentado a continuación es un ejemplo de cuando el RCX II juega el papel de receptor, esto sucede en el momento en que se encuentra en espera de una indicación del RCX I que se ha encontrado un cubo.

```
task hearing()
{
        ClearMessage();
        find_cube=0;
        while(true)
        {
                if(Message()==ARRIVE)
                {
                        find_cube=1;
                        break;
                }
        }
        stop hearing;
}
```

5.3 Diseño del algoritmo del seguidor de paredes

Para este caso el algoritmo ya esta diseñado [Ferrari, 2002], sólo se hace la adaptación para que recorra paredes del lado derecho y no del izquierdo como viene originalmente.

Al algoritmo consiste en lo siguiente:

1. Pegarse a una pared del lado derecho y seguirla.
2. Si se encuentra con una pared de frente, evitarla girando 90 grados y repetir el paso1.
3. Eventualmente el robot habrá recorrido toda la arena a través de las paredes.

El código del programa que ejecuta este comportamiento se muestra a continuación

```
task avoid()
{
	while(true)
	{
		if(RIGHT==0)
		{
		 closeCommand=COMMAND_STOP;
		 Wait(5);
		 closeCommand=COMMAND_TURN_RIGHT;
		 until(RIGHT==1);
		 closeCommand=COMMAND_STOP;
		 Wait(5);
		 closeCommand=COMMAND_NONE;
		}
		else
		{
		 closeCommand=COMMAND_NONE;
		}
	}
}
```

5.4 Implementación de la arquitectura *Subsumption*

Para implementar la arquitectura *Subsumption*, se deben implementar los siguientes métodos:

- Un *task* encargado de manejar todos los comportamientos.

```
//Lleva a cabo la coordinación de los comportamientos

int motorCommand;
task behaviors()
{
	while(true)
	{
		if(fwdCommand!=COMMAND_NONE)
			motorCommand=fwdCommand;
		if(closeCommand!=COMMAND_NONE)
			motorCommand=closeCommand;
		if(turnCommand!=COMMAND_NONE)
			motorCommand=turnCommand;
		if(sendCommand!=COMMAND_NONE)
			motorCommand=sendCommand;
		motorCrontol();
	}
}
```

- Un *task* para cada comportamiento del robot; los comportamientos fueron planteados en el diseño. A continuación se presenta el comportamiento de dar vuelta cuando se topa con el *bumper* delantero.

```
//Behavior de dar vuelta cuando se topa con el bumper  delantero
int turnCommand;

task wall()
{
	while(true)
	{
		if(FRONT==0)
		{
			turnCommand=COMMAND_STOP;
			Wait(5);

			//Indica que se ha alcanzado la posicion E1
			checkE1++;

			turnCommand=COMMAND_NONE;
		}
		else
		{
			turnCommand=COMMAND_NONE;
		}
	}
}
```

- Una función que determine los posibles movimientos del robot.

```
void motorCrontol()
{
	if(motorCommand==COMMAND_FORWARD)
	{
		OnFwd(RIGHT_M+LEFT_M);
	}
	if(motorCommand==COMMAND_REVERSE)
	{
		OnRev(RIGHT_M+LEFT_M);
	}
	if(motorCommand==COMMAND_TURN_RIGHT)
	{
		OnFwd(LEFT_M);
		OnRev(RIGHT_M);
	}
	if(motorCommand==COMMAND_TURN_LEFT)
	{
		OnRev(LEFT_M);
		OnFwd(RIGHT_M);
	}
	if(motorCommand==COMMAND_STOP)
	{
```

```
            Off(LEFT_M+RIGHT_M);
        }
        if(motorCommand==COMMAND_CORRECT)
        {
            Float(RIGHT_M);
            OnFwd(LEFT_M);
        }
}
```

Todos los *tasks* son inicializados a partir de un método *main*. El orden en que aparecen las variables que hacen referencia a los *behaviors* es de menor a mayor jerarquía, es decir, la primer variable queda suprimida por el valor de la última variable.

5.5 Diseño del algoritmo que lee la receta

La receta será leída por el robot M, el algoritmo se presenta en pseudocódigo a continuación:

1. Avanzar hacia delante hasta encontrar una línea negra.
2. Avanzar hasta encontrar una línea blanca.
3. Una vez leída la línea blanca, avanzar hasta que se lea un color distinto del blanco.
4. Una vez detectado el color, clasificarlo en negro o azul y guardarlo en la posición del arreglo correspondiente.
5. Ejecutar desde el paso 2 hasta que se tengan los 5 ingredientes.

5.6 Diseño del algoritmo que toma los cubos

El algoritmo inicia cuando se ha llegado a cualquiera de los dos repositorios, y termina una vez que se ha tomado al menos un cubo. El algoritmo se ejecuta en ambos RCX del robot del refrigerador, para denotar cada robot se utilizara RI y RII, donde RI es el encargado de las bandas y RII de la navegación.

El algoritmo queda descrito de la siguiente manera:

1. RII envía un mensaje a RI para notificar que se ha llegado al depósito.
2. RI recibe el mensaje y activa la banda que sube los cubos.
3. RII entra al depósito siguiendo una trayectoria de línea recta.
4. RII avanza un tiempo determinado hacia delante y regresa a la entrada del repositorio; este paso se ejecuta hasta que se topa con pared, avanzando cada ejecución un tiempo mayor al anterior.
5. RI monitorea si ha subido un cubo.
6. Si se ha subido un cubo, RI envía un mensaje de término al robot encargado de la navegación y ejecutar el paso 9, de lo contrario ejecutar el paso 7.
7. RII regresa a donde empieza el depósito y puede realizar los siguientes movimientos, dependiendo del estado anterior en que se encontraba:
    a. Giro a la izquierda para búsquedas a la izquierda
    b. Giro a la derecha para búsquedas por el centro.
    c. Giro a la derecha para búsquedas por la derecha.
8. RII ejecuta todos los pasos correspondientes desde el paso 4.
9. RII regresa a la línea donde inicia el repositorio y dependiendo de los movimientos que se hayan efectuado en la búsqueda, se ejecuta lo siguiente:
    a. Si el robot ha quedado girado a la izquierda, ejecutar un giro a la derecha.
    b. Si el robot ha quedado girado a la derecha, ejecutar un giro a la izquierda.

El objetivo del algoritmo es recorrer el repositorio de cubos como se muestra en la figura 5.2, y cuando se ha tomado el cubo, regresar a la posición inicial indicada como P1, los números 1, 2 y 3 es el orden en el que se ejecutan los movimientos, y las flechas indican la orientación que sigue el robot.

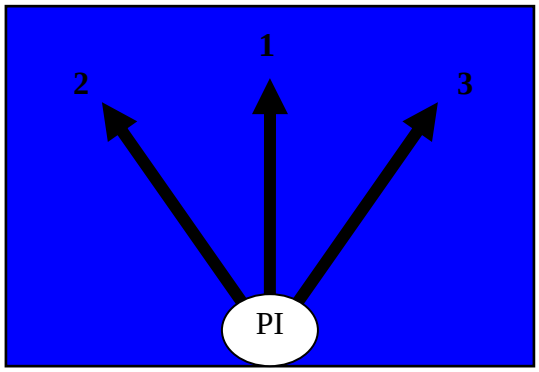

Figura 5.2 Recorrido del robot en el repositorio

5.7 Diseño de algoritmos de navegación

La navegación de un robot se relaciona con el posicionamiento del mismo. Para ello existen dos tipos de posicionamiento: absoluto y relativo. El posicionamiento relativo en *LEGO* acumula errores, lo que no nos permite utilizarlo como un posicionamiento fiable [Ferrari, 2002]. El posicionamiento absoluto se basa en el uso de señales externas que puedan orientar al robot.

Para la navegación de ambos robots, se utilizaron distintas técnicas para el posicionamiento. Se utilizaron líneas negras en el piso que sirven de referencia para movimientos efectuados por el robot, líneas negras que sigue el robot y por último algoritmos de corrección, basados en modificar la potencia de los motores.

5.8 Diseño de algoritmos de calibración

El algoritmo de calibración está orientado al uso del sensor de luz cuando las condiciones de luminosidad varíen. El objetivo del algoritmo es obtener los valores de los colores unos

minutos antes de la prueba, de tal manera que no se tenga que volver a bajar el código al RCX.

El algoritmo utiliza un sensor como acción de entrada que se activa cuando se tiene el sensor apuntando a lo que se quiere leer el algoritmo queda descrito a continuación:

1. Esperar en un ciclo infinito hasta que se active el sensor.
2. Leer el valor del sensor y asignarlo a la variable deseada.
3. Emitir un sonido para confirmar que se ha leído el color.
4. Tener un tiempo de espera con el fin de que no se ejecute el siguiente método en el lapso que el sensor es desactivado.
5. Si se ha terminado de leer las variables deseadas, terminar; de lo contrario, ejecutar desde el paso 1.

El resultado final de la programación y construcción, son robots autónomos que están orientados a resolver el problema planteado al inicio de la tesis. Para saber si han logrado los resultados deseados, se realizarán distintas pruebas en el siguiente capítulo, todas ellas orientadas a la evaluación de los robots.

5.9 Beneficios del código usado.

El código utilizado define cada uno de los movimientos del robot por separado, estos nos trae beneficios al promover la modularidad del código, por ende, nos facilita encontrar errores y cambiar algún movimiento. Otra de las ventajas es el uso de *multitasking* ya que permite que el robot realice una tarea mientras verifica otra. El uso adecuado de constantes para definir los tiempos que el robot desempeña cierta tarea nos permitió hacer un solo cambio y verlo reflejado a lo largo de todo el programa. Los demás beneficios del código propuesto se mencionan en las conclusiones del documento.