Práctica No. 3
Introducción a las Máquinas de Estados

Objetivo: Familiarizar al alumno en el conocimiento de los algoritmos de las máquinas de estados, comportamientos. Programar estos comportamientos en robots LEGO/TETRIX.

Desarrollo: Para cada uno de los siguientes apartados, realizar los diseños que se piden.

Duración: Dos semanas

1. En la figura 1 se muestra el comportamiento de un robot que evade obstáculos.

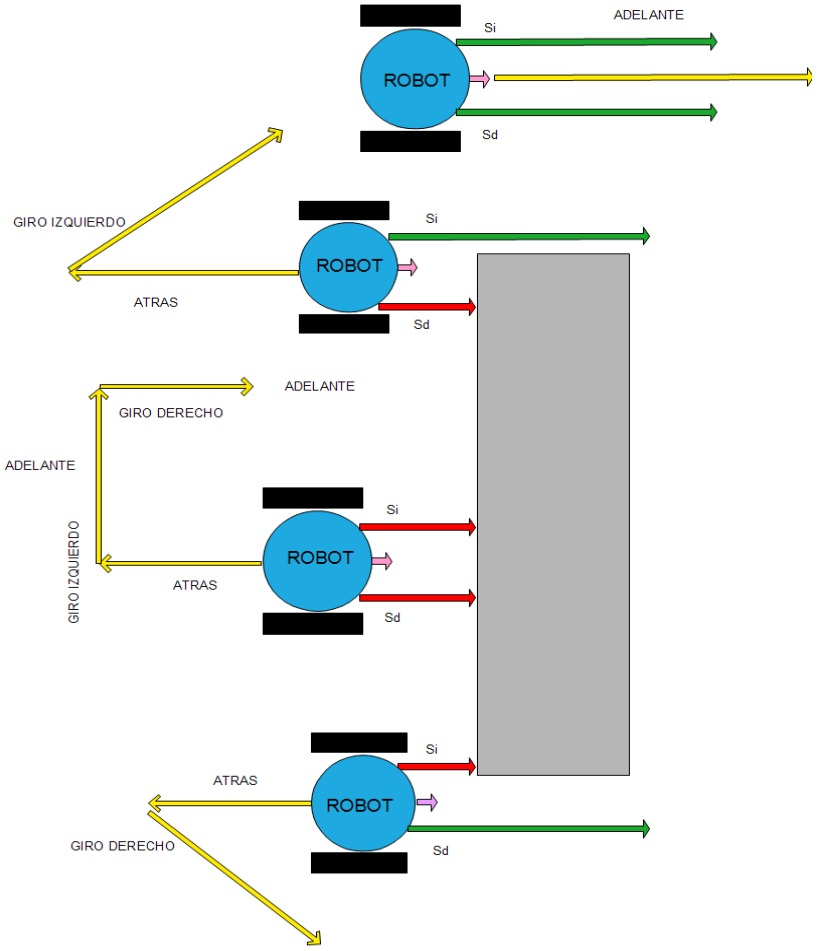

Figura 1. Robot Móvil que evade obstáculos

En la figura 2 se muestra el algoritmo o carta ASM de este robot, en éste cuando los sensores de tacto Si y Sd sensan un obstáculo sus valores son igual a uno, en caso contrario son cero.

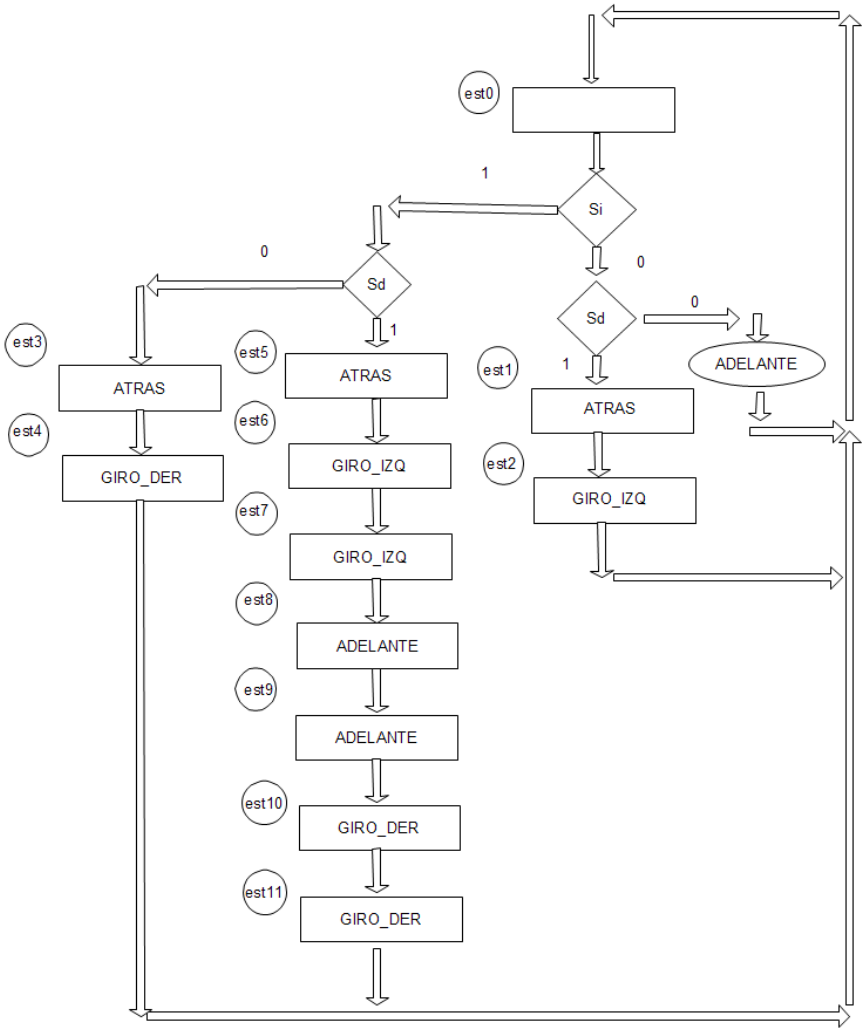

Figura 2. Algoritmo de un robot móvil que evade obstáculos

A continuación se presenta este algoritmo usando el lenguaje de programación de robotC, para un robot LEGO/TETRIX, con dos sonares.
Pruebe este código en el kit de LEGO/TETRIX, modificando, si es necesario, algunas de las constantes para que el robot tenga un movimiento de acuerdo al algoritmo de evasión de obstaculos.

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
#pragma config(Sensor, S1,     ,             sensorI2CMuxController)
#pragma config(Sensor, S2,     senD,         sensorSONAR)
#pragma config(Sensor, S3,     senI,         sensorSONAR)
#pragma config(Motor,  mtr_S1_C1_1,     motorD,        tmotorNormal, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C1_2,     motorE,        tmotorNormal, openLoop)
#pragma config(Servo,  srvo_S1_C2_1,    servo1,            tServoNone)
#pragma config(Servo,  srvo_S1_C2_2,    servo2,            tServoNone)
#pragma config(Servo,  srvo_S1_C2_3,    servo3,            tServoNone)
#pragma config(Servo,  srvo_S1_C2_4,    servo4,            tServoNone)
#pragma config(Servo,  srvo_S1_C2_5,    servo5,            tServoNone)
#pragma config(Servo,  srvo_S1_C2_6,    servo6,            tServoNone)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

/*********************************************************
 *                                                       *
 *       Obstacle Avoidance State Machine                *
 *       For  a  Robot LEGO/TETRIX and ROBOTC            *
 *                                                       *
 *       evade.c                                         *
 *                                                       *
 *       Jesus Savage                                    *
 *       Edna Marquez                                    *
 *       Lissete Castro                                  *
 *                                                       *
 *       FI-UNAM                                         *
 *                                                        *
 *                              20-4-2014                *
 *                                                       *
 *********************************************************/
#define LIMIT_Si  25 //
#define LIMIT_Sd  25 //
#define WEELS_RADIO .01
#define ROBOTS_DIAMETER .1
//#define CNT_ROTATE_LEFT (-Robots_Diameter)/(4*PI*Weels_Radio)
//#define CNT_ROTATE_RIGHT (Robots_Diameter)/(4*PI*Weels_Radio)
//#define CNT_ADVANCE 10*CNT_ROTATE_RIGHT
//#define CNT_ANGLE 1*CNT_ROTATE_RIGHT
#define POWER_LEVEL1 17
#define POWER_LEVEL2 17
#define CNT_ANGLE_RIGHT 40
#define CNT_ANGLE_LEFT 40
#define CNT_DISTANCE 8000
#define MAGNITUDE .10   // advance magnitude in meters
#define TIME_STOP 300
#ifndef PI
#define PI 3.1415926535
#endif
#define RAD_45 10 //experimental value
//#define RAD_45 (float) ((PI*45.0)/180.0); // robot's turn in radians

#define STATE_0 0
#define STATE_1 1
```

```
#define STATE_2 2
#define STATE_3 3
#define STATE_4 4
#define STATE_5 5
#define STATE_6 6
#define STATE_7 7
#define STATE_8 8
#define STATE_9 9
#define STATE_10 10
#define STATE_11 11

// Global Variables
int Si,Sd;

// It gets the sensors values
int get_sensors_values(){
 int status = 1;

 Si= SensorValue[senI];
 Sd= SensorValue[senD];

 return(status);
}


// First it rotates the robot the specified angle and then it advances the asked distance
int mv(float distance,float angle){

  int status = 1;
  int time_angle,time_distance;

  if(angle > 0){
    time_angle= CNT_ANGLE_LEFT*angle;
              motor[motorD] = -POWER_LEVEL1;
              motor[motorE] = POWER_LEVEL2;
  }
            else{
              time_angle= - CNT_ANGLE_RIGHT*angle;
              motor[motorD] = POWER_LEVEL1;
              motor[motorE] = -POWER_LEVEL2;
  }
            wait1Msec(time_angle);

            if(distance > 0){
              time_distance=  CNT_DISTANCE*distance;
              motor[motorD] = POWER_LEVEL1;
              motor[motorE] = POWER_LEVEL2;
  }
  else{
    time_distance=  -CNT_DISTANCE*distance;
    motor[motorD] = -POWER_LEVEL1;
    motor[motorE] = -POWER_LEVEL2;
  }
            wait1Msec(time_distance);

  return(status);
 }
```

```
void initialization(){

}

// The robot goes forward some distance
int forward(){
   float distance=MAGNITUDE;
   float angle= 0.0;
   int status;

   nxtDisplayTextLine(3, "Action FORWARD ");
   status=mv(distance,angle);

   return status;

}

// The robot goes backward some distance
int backward(){
   float distance=-MAGNITUDE;
   float angle= 0.0; //before float angle= 0.0;
   int status;

   nxtDisplayTextLine(3, "Action BACKWARD ");
   status=mv(distance,angle);

   return status;
}

// The robot turns to the left 45 degrees (positive angle)
int turn_left(){
   float distance=0.0;//float distance=0.0f;
   float angle=RAD_45;
   int status;

   nxtDisplayTextLine(3, "Action TURN_LEFT ");
   status=mv(distance,angle);

   return status;
}

// The robot turns to the right 45 degrees (nagative angle)
int turn_right(){
   float distance=0.0;//before: float distance=0.0f;
   float angle=-RAD_45;
   int status;

   nxtDisplayTextLine(3, "Action TURN_RIGHT ");
   status=mv(distance,angle);

   return status;
}

// The robots stops
int stop(){

   int status=1;
   int time_stop;
```

```
    nxtDisplayTextLine(3, "Action STOP ");
    motor[motorD] = 0;
            motor[motorE] = 0;

    time_stop= TIME_STOP;
    wait1Msec(time_stop);
    return status;

}

// Algorithm to avoid obstacles
void evade_obstacles(){

  int state = STATE_0;
  int i=1;


  while(true){

    // it gets the sensors inputs
    get_sensors_values();
    nxtDisplayTextLine(0, "iteration %d",i);
    nxtDisplayTextLine(1, "state %d",state );
    nxtDisplayTextLine(2, "sd %d si %d", Sd,Si);

    switch(state){

     case STATE_0:
         if(Si>=LIMIT_Si && Sd>=LIMIT_Sd) {
             forward();
             state=STATE_0;
         }
                 else{
                   stop();

               if(Si>=LIMIT_Si && Sd<LIMIT_Sd) state = STATE_1;
               else if(Si<LIMIT_Si && Sd>=LIMIT_Sd) state = STATE_3;
               else if(Si<LIMIT_Si && Sd<LIMIT_Sd) state = STATE_5;

                  }
        break;

     case STATE_1:
         backward();
         state = STATE_2;
         break;

     case STATE_2:
         turn_left();
         state = STATE_0;
         break;

     case STATE_3:
         backward();
         state = STATE_4;
         break;

     case STATE_4:
```

```
        turn_right();
        state = STATE_0;
        break;

    case STATE_5:
        backward();
        state = STATE_6;
        break;

    case STATE_6:
        turn_left();
        state = STATE_7;
        break;

    case STATE_7:
        turn_left();
        state = STATE_8;
        break;

    case STATE_8:
        forward();
        state = STATE_9;
        break;

    case STATE_9:
        forward();
        state = STATE_10;
        break;

    case STATE_10:
        turn_right();
        state = STATE_11;
        break;

    case STATE_11:
        turn_right();
        state = STATE_0;
        break;

   }

   nxtDisplayTextLine(4, "next state %d",state );
   i++;

  }
}


// Main program
task main (){

            // Initialization
            initialization();

            // Algorithm to avoid obstacles
            evade_obstacles();

}
```