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 con el sistema ROS y robots TURTLE BOT

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.

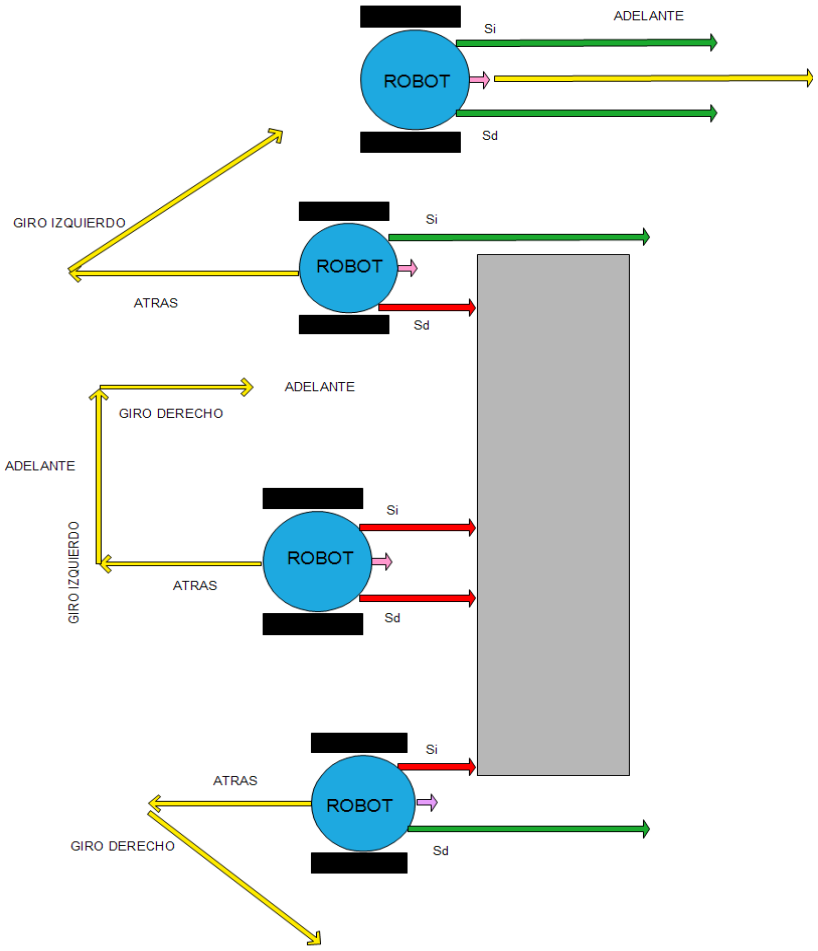

Figura 1. Robot Móvil que evade obstáculos

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.

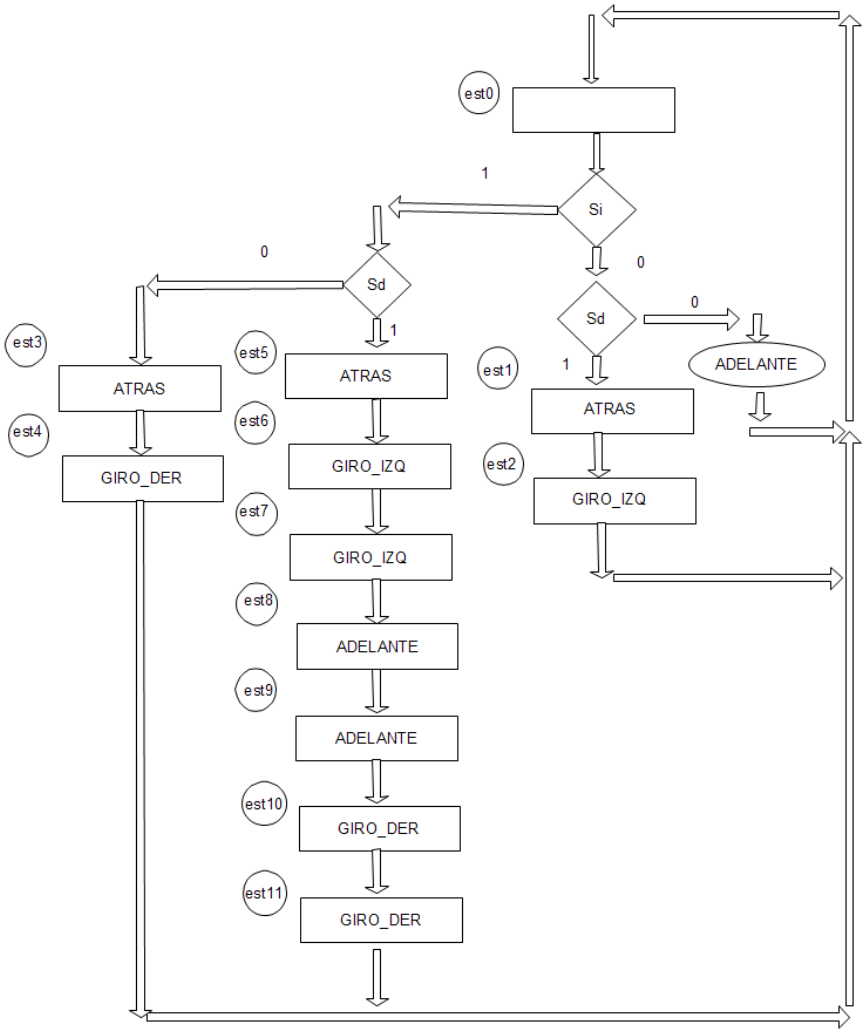

Figura 2. Algoritmo de un robot móvil que evade obstáculos

A continuación se presenta este algoritmo usando el lenguaje de programación C++, evade.cpp y ros.h, para un robot TURTLE BOT, usando lecturas del laser Hokuyo.
Pruebe este código en el robot con ROS, modificando, si es necesario, algunas de las constantes para que el robot tenga un movimiento de acuerdo al algoritmo de evasión de obstaculos.

Código evade.cpp

```
/*************************************************************
 *                                                           *
 *       Obstacle Avoidance State Machine                    *
 *       evade.cpp                                           *
 *                                                           *
 *       Jesus Savage                                        *
 *       Jaime Marquez                                       *
 *       Mauricio Matamoros                                  *
 *                                                           *
 *       FI-UNAM                                             *
 *                                                           *
 *                              18-3-2014                    *
 *                                                           *
 *************************************************************/

#include "ros.h"

#define MAGNITUDE 1   // advance magnitude in meters
#define RAD_45 1.7; // 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
bool wall_;
float Si,Sd;


void callbackbumper(const create_node::TurtlebotSensorState::ConstPtr& msg){
   wall_= msg->bumps_wheeldrops;
}

void callbacklaser(const sensor_msgs::LaserScan::ConstPtr& msg){
  Sd=msg->ranges[138]+msg->ranges[139]+msg->ranges[140]+msg->ranges[141];
  Si=msg->ranges[370]+msg->ranges[371]+msg->ranges[372]+msg->ranges[373];
  Sd/=4;
  Si/=4;
}
```

```
// It reads from a node the laser readings that represent the average left and right side
int get_laser_values(){
   int status = 1;
   ROS_INFO_STREAM("Laser");
   ros::NodeHandle nh;
   ros::Subscriber laser= nh.subscribe<sensor_msgs::LaserScan>("/scan", 1000, callbacklaser);
   ros::spin();
   return(status);
}


int get_bumper_values(){
   int status = 1;
   ROS_INFO_STREAM("Bumper");
   ros::NodeHandle nh;
   ros::Subscriberwall=nh.subscribe<create_node::TurtlebotSensorState>("/mobile_base/sensors/cor
e", 1000, callbackbumper);
   ros::spin();
   return(status);
}


// The robot goes forward some distance
int forward(){
   float distance=MAGNITUDE;
   float angle= 0.0f;
   int status;
   status=mv(distance,angle);
   printf(" Action FORWARD ");
   return status;

}


// The robot goes backward some distance
int backward(){
   float distance=-MAGNITUDE;
   float angle= 0.0f;
   int status;
   status=mv(distance,angle);
   printf(" Action BACKWARD ");
   return status;
}


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

```
    int status;
    status=mv(distance,angle);
    printf(" Action TURN_LEFT ");
    return status;
}


// The robot turns to the right 45 degrees (nagative angle)
int turn_right(){
    float distance=0.0f;
    float angle=-RAD_45;
    int status;
    status=mv(distance,angle);
    printf(" Action TURN_RIGHT ");
    return status;
}


// The robots stops
int stop(){
    float distance=0.0f;
    float angle=0.0f;
    int status;
    status=mv(distance,angle);
    printf(" Action STOP ");
    return status;

}



// Algorithm to avoid obstacles
void evade_obstacles(){
    int state = STATE_0;
    int status;


    while(ros::ok()){
    // it gets the sensors inputs
        switch(state){

            case STATE_0:
                if(Si>LIMIT_Si && Sd>LIMIT_Sd)
                    forward();
                else{
                    if(Si>LIMIT_Si && Sd<LIMIT_Sd)
                        state = STATE_1;
                    if(Si<LIMIT_Si && Sd>LIMIT_Sd)
                        state = STATE_3;
                    if(Si<LIMIT_Si && Sd<LIMIT_Sd)
                        state = STATE_5;
```

```
        stop();
    }
    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;

      }
      printf(" next state %d\n",state);
   }
}



// Main program
int main(int argc, char *argv[]){
   ROS_INFO_STREAM("main");

   // Initialization
   initialization(argc,argv);
   boost::thread t1(&get_laser_values);

   // Algorithm to avoid obstacles
   evade_obstacles();
   t1.join();
}
```

Código ros.h

```
/*****************************************************************************
 *                          ros.h                                            *
 *                          ROS Basic Functions                              *
 *                                                                           *
 *                                                                           *
 *      Jesus Savage                                                         *
 *      Jaime Marquez                                                        *
 *      Mauricio Matamoros                                                   *
 *      FI-UNAM                                                              *
 *                                                                           *
 *                          18-3-2014                                        *
 *                                                                           *
 *                                                                           *
 *****************************************************************************/
#include <geometry_msgs/Twist.h>
#include <stdio.h>
#include "ros/ros.h"
#include <create_node/TurtlebotSensorState.h>
#include <boost/thread.hpp>
#include <sensor_msgs/LaserScan.h>

#define LIMIT_Si (float)0.3
#define LIMIT_Sd (float)0.3

// It publish the speeds of the left and right weels
// First it rotates the robot the specified angle and then it advances the asked distance
int mv(float distance,float angle){


        int status = 1;
        ros::NodeHandle nh;
        ros::Publisher cmd_vel_pub=nh.advertise<geometry_msgs::Twist>(
                                        "/mobile_base/commands/velocity", 1000);
        ros::Rate loop_rate(1);
        geometry_msgs::Twist base_cmd;

        base_cmd.linear.x = base_cmd.linear.y = base_cmd.angular.z = 0;

        if(angle!=0){
                base_cmd.angular.z=angle;
                cmd_vel_pub.publish(base_cmd);
                ros::spinOnce();
```

```
                loop_rate.sleep();
                ROS_INFO_STREAM("angular");

        }else{
                base_cmd.linear.x =distance;
                cmd_vel_pub.publish(base_cmd);
                ros::spinOnce();
                loop_rate.sleep();
                ROS_INFO_STREAM("linear");
        }

        return(status);
}

void initialization(int argc, char **argv){
        ROS_INFO_STREAM("inicializacion");
        ros::init(argc, argv, "turtle_prac");
}
```