Memoria Práctica 1 TSI: Random Walk

Javier Béjar Méndez

1 – Objetivo

El objetivo de esta entrega consiste en conocer los aspectos básicos de la implementación de software para controlar un robot (simulado) usando ROS. La tarea principal es crear un nodo ROS que dirigirá el robot por el entorno simulado de Gazebo siguiendo un algoritmo básico de navegación aleatoria o deambulación, similar al que sigue una aspiradora robot. El robot se moverá hacia adelante hasta que se encuentre cerca de un obstáculo, entonces rotará por un tiempo aleatorio y volverá a moverse hacia adelante. La dirección de la rotación estará determinada por la zona en la que el robot vea que hay menos obstáculos.

2 – Solución

Primero generamos el paquete Random_Walk, con las siguientes dependencias: *std_msgs*, *rospy*, *roscpp*. Generamos el nodo Rwalk, cuyo comportamiento es el descrito en la sección anterior, se suscribe al nodo *laserSub* para leer la información del sensor laser simulado del robot. Publica en el topic *commandPub* para rotar y mover al robot.

El archivo de cabecera del módulo Rwalk es el siguiente:

```
#ifndef RANDOM_WALK_H_
#define RANDOM_WALK_H_
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"

class Rwalk {
public:
    // Tunable parameters
    const static double FORWARD_SPEED = 0.3;
    const static double ANGULAR_SPEED = 0.2;
    const static double MIN_SCAN_ANGLE = -30.0/180*M_PI;
    const static double MAX_SCAN_ANGLE = +30.0/180*M_PI;
    const static float MIN_DIST_FROM_OBSTACLE = 0.8; // Should be smaller than sensor_

    Rwalk();
    void startMoving();

private:
    ros::NodeHandle node;
    ros::Publisher commandPub; // Publisher to the robot's velocity command topic
    ros::Subscriber laserSub; // Subscriber to the robot's laser scan topic
    bool keepMoving; // Indicates whether the robot should continue moving
    int dirRotate;
    bool rotating;

    void moveForward();
    void rotate(int dir);
    void scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan);
};

#endif
```

Illustration 1: Archivo Rwalk.h

Añadimos el macro *ANGULAR_SPEED* que define la velocidad de rotación del robot, las variables *dirRotate* que indica la dirección del giro (-1 antihorario, 1 horario) y *rotating* que indica si el robot está girando, y el método *rotate(dir)* que envía el mensaje con la dirección de rotación indicada sobre el eje z, como podemos observar en la siguiente figura:

```
void Rwalk::rotate(int dir)
{


    geometry_msgs::Twist msg;
    msg.angular.z = ANGULAR_SPEED * dir;
    commandPub.publish(msg);
}
```

Illustration 2: Función rotate

El robot cuando encuentra un obstáculo con una distancia menor a la indicada en *MIN_DIST_FROM_OBSTACLE* se para y analiza todas las mediciones del sensor laser para girar en el sentido contrario a donde se encuentre el obstaculo mas cercano, es decir, tomando como ángulos mínimo y máximo, -30 y 30 grados, si el objeto mas cercano se encuentra en un ángulo menor que 0 el robot rota en el sentido de las agujas del reloj, y si es mayor o igual en el sentido contrario, el codigo que describe dicho comportamiento es el siguiente:

```
void Rwalk::scanCallback(const sensor_msgs::LaserScan::ConstPtr& scan)
{
    bool isObstacleInFront = false;

    // Find the closest range between the defined minimum and maximum angles
    int minIndex = ceil((MIN_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);
    int maxIndex = floor((MAX_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);

    float currAngle = scan->angle_min;
    float minCloseAngle;
    double mindist = MIN_DIST_FROM_OBSTACLE;

    //Recorre todas las mediciones para guardar el angulo de la medicion mas cercana
    for (int currIndex = minIndex + 1; currIndex <= maxIndex; currIndex++)
    {
        currAngle += scan->angle_increment; //Angulo de la medicion actual
        if (scan->ranges[currIndex] < mindist)
        {
            if(!isObstacleInFront) //Solo manda stop la primera vez
            {
                isObstacleInFront = true;
            }
            mindist = scan->ranges[currIndex]; //distancia al bjeto mas cercano
            minCloseAngle = currAngle; //Angulo del objeto mas cercano
        }
    }

    //Si encuentra obstaculo y no esta rotando
    if(isObstacleInFront && !rotating)
    {
        ROS_INFO("Stop!");
        keepMoving = false; //Se para
        ROS_INFO("minCloseAngle: %f", minCloseAngle);
        dirRotate = (minCloseAngle < 0 ? 1 : -1); //Direccion opuesta al objeto mas cercano
    }
}
```

Illustration 3: Función scanCallBack

Solo queda describir el bucle general del robot y el comportamiento aleatorio en cuanto a la duración del giro, básicamente el robot avanza hacia delante cuando no hay obstáculos, si encuentra un obstáculo se para y gira durante un tiempo aleatorio (0 a 7 segundos), para volver a avanzar, hasta que el usuario cierre la aplicación. En la siguiente ilustración podemos observar el bucle principal:

```
void Rwalk::startMoving()
{
    ros::Rate rate(10);
    ROS_INFO("Start moving");
    double begin;
    int tRot;
    // Keep spinning loop until user presses Ctrl+C or the robot got too
    while (ros::ok()) {
        while(keepMoving)
        {
            moveForward();
            ros::spinOnce(); // Need to call this function often to allow
            rate.sleep();
        }
        tRot = (rand() % 8); //Tiempo aleatorio de rotacion [0, 7]
        begin = ros::Time::now().toSec() + tRot;
        if(dirRotate > 0)
            ROS_INFO("Turning left for %d secs", tRot);
        else ROS_INFO("Turning right for %d secs", tRot);
        while(begin - ros::Time::now().toSec() > 0)
        {
            rotate(dirRotate);
            ros::spinOnce(); // Need to call this function often to allow
            rate.sleep();
        }
        rotating = false; //Para de rotar
        keepMoving = true;
    }
}
```

Illustration 4: Función startMoving