Memoria Practica 1

Antonio Manuel Milán Jiménez Grupo 3

Explicación del problema:

Esta practica consiste en programar un comportamiento para Turtlebot en su desplazamiento. Este desplazamiento tiene que consistir en que Turtlebot avance hacia delante hasta que se encuentre con un objeto.

Cuando se encuentra con un objeto tiene que decidir hacia qué lado girar en función del lado en el que perciba menos objetos. Una vez que ha decidido el lado hacia el que tiene que girar, gira en esa dirección un número de grados aleatorio. Entonces continuaría avanzando y se repetiría el proceso.

Solución del problema:

Para resolver este problema se ha cogido como base el programa de “Stopper” que hacía que el robot se parase al encontrar un objeto en su campo de visión y a una distancia mínima de él.

Partiendo de este programa, ahora al encontrarse un objeto delante, analiza en cual de las dos mitades de su campo de visión hay menos objetos, tal y como se muestra a continuación:

```
int minIndex = ceil((MIN_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);
int maxIndex = floor((MAX_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);
medio_angulo = (minIndex+1+maxIndex)/2;

for (int currIndex = minIndex + 1; currIndex <= medio_angulo; currIndex++) {
    if (scan->ranges[currIndex] < MIN_DIST_FROM_OBSTACLE) {
      contador1++;
    }
}

for (int currIndex = medio_angulo + 1; currIndex <= maxIndex; currIndex++) {
    if (scan->ranges[currIndex] < MIN_DIST_FROM_OBSTACLE) {
      contador2++;
    }
}

if(contador1 < contador2)
  return 1;
else
  return 2;
```

Así, tendríamos dos contadores donde almacenamos el número de veces que se ha encontrado objetos en ambas mitades del campo de visión y se girará hacia el lado donde haya menos, ya que en principio por ese lado se encontrará menos obstáculos.

Una vez que ha decidido hacia que lado tiene que girar, tiene que decidir cuánto va a girar. Esta decisión es aleatoria dentro de un rango de 10 a 100 grados. La principal razón por la que es aleatoria es para evitar que entre en un ciclo haciendo siempre el mismo recorrido si le pusiésemos

que siempre girase un determinado ángulo. Aquí vemos como se elige aleatoriamente cuantos grados girar:

```
giro = objetosDetectados(scan);

int r = rand()%100 +10;

if(giro == 1){
    girarDcha1(r);
}else
    girarIzq1(r);
```

Y aquí tenemos como se realiza un giro de los grados aleatorios que se indican, tanto a la derecha como a la izquierda. Controlando el “rate” o velocidad de ciclo, la velocidad angular y usando “r.sleep()”, podemos hacer que se giren esos grados tal y como se muestra a continuación:

```
void RandomWalk::girarIzq1(int x){
  geometry_msgs::Twist msg;
  ros::Rate r(1); //1 ciclo por s

  int i=0;
  while(i!=1){
      girarIzq2(x);
      r.sleep();
      msg.angular.z = 0;
      commandPub.publish(msg);
      i=1;
  }
}

void RandomWalk::girarIzq2(int x) {
    geometry_msgs::Twist msg; // The default constructor wi
    msg.angular.z = x;
    commandPub.publish(msg);
    ROS_INFO("Girando");
}
```

Finalmente se ha hecho una pequeña modificación en su visión, reduciendo el ángulo de visión que tiene en cuenta cuando está pendiente de que se encuentre un obstáculo delante. Esto se hace para evitar que Turtlebot se crea que se va a encontrar con un obstáculo delante cuando realmente esta suficientemente al lado para poder pasar sin tener que girar:

```
int minIndex = ceil((MIN_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);
int maxIndex = floor((MAX_SCAN_ANGLE - scan->angle_min) / scan->angle_increment);
minIndex = minIndex + 1;
maxIndex = maxIndex-1;
```