Práctica 3
-
Informe

Julen D. y Julen F.

3.1. Calibración de sensores de choque y caída de ruedas (Bumps & WheelDrops)

3.1 Calibración de sensores de choque y caída de ruedas (Bumps & WheelDrops):

- Diseñar un programa que reaccione a los cambios en los Bumpers emitiendo una señal audible diferente
- Diseñar otro programa que reaccione a los Wheeldrops, encendiendo alguno de los leds del robot

Valores que devuelve el sensor BUMPERS_AND_WHEELDROPS:

- Nada = 0
- BumperRight = 1
- BumperLeft = 2
- RightWheelDrop = 4
- LeftWheelDrop = 8
- FrontWheelDrop = 16

Estos valores los devuelve cada sensor mencionado por separado. Es decir, si se levantan las tres ruedas, el valor devuelto será 4+8+16 = 28.

→ Programa que reacciona a los bumpers emitiendo señales audibles

Para detectar si el bumper ha sido accionado, se comprueba si el valor devuelto por el sensor BUMPERS_AND_WHEELDROPS está entre 1 y 3. (BumperRight, BumperLeft o BumperRight+BumperLeft). Para el ejercicio, se han definido tres tonos diferentes que reaccionan a los sensores.

Código:

```
int i= 0;
cout << "Comprobar el estado del sensor choque en 2 seg" << endl;
delay(2000);
robot.song(1,2,"64 10");
robot.song(2,2,"80 10");
robot.song(3,2,"100 10");
while(i < 30){
    sensor_value = robot.updateSensor(iRobotSensors::BUMPERS_AND_WHEELDROPS);
    cout << "Valor: " << +sensor_value << endl;
    if (sensor_value != 0 && sensor_value < 4) {
        robot.playSong(sensor_value);
    }
    i++;
    delay(1000);
}
```

→ WheelDrops

Para detectar si alguna rueda está caída, se comprueba si el valor devuelto por el sensor BUMPERS_AND_WHEELDROPS coincide con uno de los valores arriba mencionados. Para el ejercicio, se han escogido los tres leds que el iRobot tiene junto a los botones.

Código:

```
int i= 0;
cout << "Comprobar el estado de caida de ruedas en 2 seg" << endl;
delay(2000);
robot.song(1,2,"64 10");
robot.song(2,2,"80 10");
robot.song(3,2,"100 10");
while(i < 30){
    sensor_value = robot.updateSensor(iRobotSensors::BUMPERS_AND_WHEELDROPS);
    cout << "Valor: " << +sensor_value << endl;
    if(sensor_value == 4) {
        robot.leds(0, 0, 255);
    } else if (sensor_value == 8) {
        robot.leds(2, 0, 255);
    } else if (sensor_value == 16) {
        robot.leds(8, 0, 255);
    } else{
        robot.leds(0, 0, 0);
    }
    i++;
    delay(1000);
}
```

3.2. Calibración de sensores de barranco (Cliffs Signal)

3.2 Calibración de sensores de barranco (Cliffs Signal):

- Diseñar un programa capaz de recoger 5 muestras de cada sensor de barranco sobre la cinta aislante negra y sobre el suelo del laboratorio.
- Para cada sensor calcular los valores medios de lectura de cinta y suelo.
- Definir un umbral de suelo y un umbral de cinta que permitan diferenciar el suelo y la cinta aislante.

Código:

```
int i = 0, media = 0;

while(i<5) {
    sensor_value = robot.updateSensor(iRobotSensors::CLIFFLEFTSIGNAL);
    cout << "Valor LEFT: " << +sensor_value << endl;
    i++;
    media += sensor_value;
    delay(1000);
}
cout << "Media LEFT: " << + (media/5) << endl;
media = 0;
i = 0;
delay(3000);
while(i<5) {
    sensor_value = robot.updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL);
    cout << "Valor FRONT LEFT: " << +sensor_value << endl;
    i++;
    media += sensor_value;
    delay(1000);
}
cout << "Media FRONT LEFT: " << + (media/5) << endl;
media = 0;
i = 0;
delay(3000);
while(i<5) {
    sensor_value = robot.updateSensor(iRobotSensors::CLIFFFRONTRIGHTSIGNAL);
    cout << "Valor FRONT RIGHT: " << +sensor_value << endl;
    i++;
    media += sensor_value;
    delay(1000);
}
cout << "Media FRONT RIGHT: " << + (media/5) << endl;
media = 0;
i = 0;
delay(3000);
while(i<5) {
    sensor_value = robot.updateSensor(iRobotSensors::CLIFFRIGHTSIGNAL);
    cout << "Valor RIGHT: " << +sensor_value << endl;
    i++;
    media += sensor_value;
    delay(1000);
}
cout << "Media RIGHT: " << + (media/5) << endl;
```

Por cada sensor se han repetido 5 veces la operación de lectura de baldosa, y se han apuntado las medias en la tabla siguiente. Se ha repetido el proceso para la lectura de cinta.

→ Muestras de los sensores - Medias:

CLIFF	BALDOSA	CINTA AISLANTE
LEFT	1327	232
FRONT LEFT	1496	316
FRONT RIGHT	1170	220
RIGHT	1836	412

Como los valores observados durante las ejecuciones son distantes unos de otros, se ha decidido que para los valores de lectura de baldosa, se obtengan los valores a partir de un mínimo. En cambio, para los valores de lectura de cinta, se procesarán a partir de un máximo. Estos valores se han decidido a partir de observar los valores sueltos de lectura devueltos por los sensores durante la ejecución.

→ Umbral de suelo y umbral de cinta de cada sensor

CLIFF	Valor	BALDOSA	CINTA AISLANTE
LEFT	min	1000	-
	MAX	-	450
FRONT LEFT	min	1100	-
	MAX	-	600
FRONT RIGHT	min	900	-
	MAX	-	500
RIGHT	min	1500	-
	MAX	-	650

3.3. Calibración del sensor de distancia a la pared (Wall)

3.3 Calibración del sensor de distancia a la pared (Wall):

- Recoger 10 muestras del sensor de pared variando la distancia. Encontrar el rango de distancias en las que el sensor devuelve 1 (wall seen).

Código:

```
int i = 0;
while(i < 10) {
    sensor_value = robot.updateSensor(iRobotSensors::WALL);
    cout << "Valor WALL: " << +sensor_value << endl;
    i++;
    delay(1000);
}
```

Para esta práctica, se ha puesto el robot a varias distancias, y a cada lectura, se acercaba o se alejaba el robot de la pared, dependiendo de si el sensor devolvía un valor u otro.

Muestras (en cm):

#	1	2	3	4	5	6	7	8	9	10
Dist.	5	4	3	3,2	3,7	3,3	3,6	3,5	3,5	3,6
Valor	0	0	1	1	0	1	0	1	1	0

3.4. Calibración de sensores de distancia para Dead-Reckoning

3.4 Calibración de sensores de distancia para Dead-Reckoning:

- Diseñar un programa que haga avanzar el robot recorriendo un cuadrado usando los sensores de distancia y ángulo. El robot terminará en la posición inicial.
  Antes de empezar, el programa debe pedir al usuario el lado del cuadrado en centímetros y la dirección de los giros (derecha/ horario/ dextrógiro o izquierda/ antihorario/ levógiro).
  El programa debe compensar los errores para que el robot acabe exactamente en el mismo punto y con la misma orientación, para cualquier tamaño del cuadrado.
  Al terminar hacer una demostración ante el profesor.

Código:

```
int velAvance = 150;
int velGiro = 150;
int distancia = atoi(argv[1]);
char* direccion = argv[2];    //d = derecha; i = izquierda

int i = 0;
while (i < 4) {   //4 giros
      //waitDistance() es crítico; se intentará hacerlo de otro modo
      robot.driveDirect(velAvance, velAvance); //avanzar
      int dr = 0; //distancia recorrida
      while (dr < distancia) { //hasta recorrer la distancia...
            dr += robot.updateSensor(iRobotSensors::DISTANCE);
            cout << "Valor DISTANCE: " << +dr << endl;
      }
      int ar = 0;
      int ang = 0;
      if(strcmp(direccion,"d")==0){ //derecha
            robot.driveDirect(-velGiro, velGiro); //girar hacia la derecha
            while (ar < 90) { //hasta girar del todo
                  //65536 = 2^16 (16 bits); Al girar a la derecha, el sensor de
angle devuelve 65534
                  ang = robot.updateSensor(iRobotSensors::ANGLE);
                  ar += (65536 - ang);
                  cout << "Valor acumulado: " << +ar << endl;
            }
      } else { //izquierda
            robot.driveDirect(velGiro, -velGiro); //girar hacia la izquierda
            while (ar < 90) { //hasta girar del todo
                  ar += robot.updateSensor(iRobotSensors::ANGLE);
                  cout << "Valor ANGLE: " << +ar << endl;
            }
      }

      i++;
}

robot.driveDirect(0, 0); //parar
```

Después de varias ejecuciones, hemos determinado que 150mm/s es una velocidad óptima tanto para avanzar como para girar. Si se hace más despacio, el robot se “adormece” y no cumple con lo pedido; si se hace más rápido, el robot no tiene tiempo de reaccionar y no cumple con lo exigido. Además, esta velocidad da unos resultados aceptables en cuanto a la posición final del robot.