Manual del Robot Ciego

Quistian Navarro, Juan Luis
A341807@alumnos.uaslp.mx

Universidad Autónoma de San Luis Potosí Ing. Sistemas Inteligentes, Gen. 2021
Programación de Robots, Grupo 281701

Abstract

Este manual documenta el desarrollo y funcionamiento de un robot móvil Khepera IV que sigue la regla de la mano derecha para navegar a través de un laberinto. Se incluyen el objetivo del proyecto, una descripción detallada del enfoque utilizado, y el código fuente del controlador documentado.

1 Objetivo del Proyecto

El objetivo del proyecto es programar un robot móvil Khepera IV para que navegue a través de un laberinto utilizando la regla de la mano derecha. Esto implica que el robot debe usar sus sensores de proximidad para evitar colisiones y siempre debe seguir la pared más cercana a su derecha para encontrar la salida del laberinto.

2 Descripción del Enfoque

Para alcanzar el objetivo, se siguió un enfoque iterativo y experimental, que se puede resumir en los siguientes pasos:

1. **Configuración del Entorno**: Se configuró un entorno de simulación en Webots con un laberinto que simula las especificaciones del escenario del parcial 2. El laberinto incluye paredes y un piso con texturas específicas.

2. **Programación de Sensores**: El robot Khepera IV fue equipado con sensores ultrasónicos frontales y laterales para detectar la distancia a las paredes. Los datos de estos sensores son cruciales para la navegación del robot.

3. **Filtro de Kalman**: Para mejorar la precisión de las lecturas de los sensores, se implementó un Filtro de Kalman. Este filtro ayuda a suavizar las lecturas y reducir el ruido, proporcionando mediciones más fiables.

4. **Implementación de la Regla de la Mano Derecha**: Se programó el robot para seguir la regla de la mano derecha. Esto implica que el robot sigue la pared derecha para poder terminar el laberinto.

3 Roboto Khepera IV

Este robot tiene 5 sensores ultrasónicos. Nosotros sólo utilizamos 2 sensores ultrasónicos. el frontal y el derecho. Además se añadió un inertial unit encima del robot para lograr hacer las rotaciones de 90°.

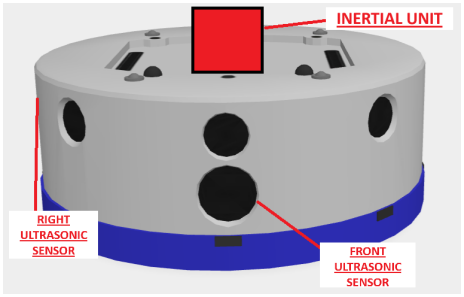

Figure 1: Esquema del robot utilizado

4 Laberinto

EL laberinto, es el usando en el parcial 2, pero se añadieron los requerimientos para el escenario que pide este proyecto. las paredes se hicieron dobles de altura siendo la pared blanca la que cumple el requerimiento actual. Se añadieron 2 objetos compuestos de Webots, la planta y una roca así como otro barril.

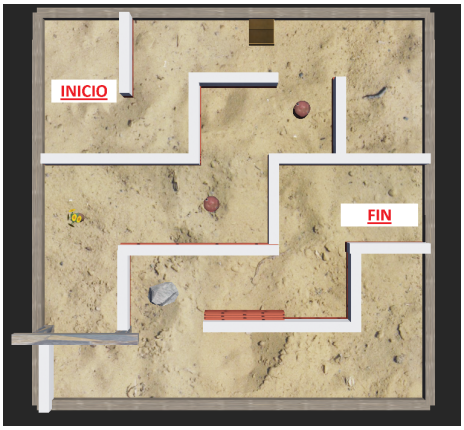

Figure 2: Esquema del laberinto utilizado en las pruebas.

5 Código Fuente Documentado

Librarías utilizadas en el controlador.

```
#include <webots/robot.h>
#include <webots/motor.h>
#include <webots/distance_sensor.h>
#include <webots/inertial_unit.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
```

Definimos las constantes.

```
#define TIME_STEP 32
#define MAX_SPEED 10
#define TURN_SPEED_FACTOR 0.3
#define DISTANCE_THRESHOLD 0.3
```

Definir matrices y variables para el filtro Kalman del sensor frontal

```
  float x_front_k[2] = {0, 0};                          // vector de estado: [distancia
    , velocidad].
  float P_front_k[2][2] = {{1, 0}, {0, 1}};             // Matriz de covarianza
  float F[2][2] = {{1, TIME_STEP / 1000.0}, {0, 1}}; // Matriz de transicion de
    estado
  float Q[2][2] = {{1e-4, 0}, {0, 1e-4}};               // Matriz de covarianza del
    ruido del proceso
  float H[2] = {1, 0};                                  // Matriz de observacion
  float R = 0.05;                                       // Matriz de covarianza del
    ruido de medicion
  float I[2][2] = {{1, 0}, {0, 1}};                     // Matriz identidad
```

Definir matrices y variables para el filtro Kalman del sensor derecho

```
  float x_right_k[2] = {0, 0};                 // vector de estado: [distancia,
    velocidad].
  float P_right_k[2][2] = {{1, 0}, {0, 1}}; // Matriz de covarianza
```

Predicción de kalman

Filtro de kalman, basado en la Implementación del usurario de github @GogiPuttar (el lo tenia en python, yo lo adapte al proyecto y cambie a C).

```
  void kalman_predict(float x_k[], float P_k[][2])
{
  // Estimacion del estado predicho
  float x_k_1[2];
  x_k_1[0] = F[0][0] * x_k[0] + F[0][1] * x_k[1];
  x_k_1[1] = F[1][0] * x_k[0] + F[1][1] * x_k[1];

  // Estimacion de la covarianza predicha
  float P_k_1[2][2];
  for (int i = 0; i < 2; i++)
  {
    for (int j = 0; j < 2; j++)
    {
```

```
      P_k_1[i][j] = F[i][0] * P_k[0][j] + F[i][1] * P_k[1][j] + Q[i][j];
    }
  }

  // Actualizar estado y covarianza
  x_k[0] = x_k_1[0];
  x_k[1] = x_k_1[1];
  for (int i = 0; i < 2; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      P_k[i][j] = P_k_1[i][j];
    }
```

Actualización de kalman

```
  void kalman_update(float x_k[], float P_k[][2], float z)
{
  // Innovacion o residual de medicion
  float y_k = z - (H[0] * x_k[0] + H[1] * x_k[1]);

  // Covarianza de la innovacion
  float S_k = H[0] * P_k[0][0] * H[0] + H[0] * P_k[0][1] * H[1] + H[1] * P_k[1][0] *
     H[0] + H[1] * P_k[1][1] * H[1] + R;

  // Ganancia de Kalman
  float K_k[2];
  K_k[0] = (P_k[0][0] * H[0] + P_k[0][1] * H[1]) / S_k;
  K_k[1] = (P_k[1][0] * H[0] + P_k[1][1] * H[1]) / S_k;

  // Estado actualizado estimado
  x_k[0] = x_k[0] + K_k[0] * y_k;
  x_k[1] = x_k[1] + K_k[1] * y_k;

  // Estimacion de la covarianza actualizada
  float P_k_1[2][2];
  for (int i = 0; i < 2; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      P_k_1[i][j] = (I[i][j] - K_k[i] * H[j]) * P_k[i][j];
    }
  }
  for (int i = 0; i < 2; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      P_k[i][j] = P_k_1[i][j];
    }
  }
}
```

Girar el robot 90°

Referencia al usuario de github @azmonreal (tenía la implementación en python yo adapte a C).

```
  void turn(int dir, WbDeviceTag left_motor, WbDeviceTag right_motor, WbDeviceTag iu
    )
{
```

```
  // Obtener el yaw actual
  const double *rpy = wb_inertial_unit_get_roll_pitch_yaw(iu);
  double c_yaw = rpy[2];
  c_yaw = round(c_yaw / (M_PI / 2)) * M_PI / 2;
  double target_yaw = fmod((c_yaw + (M_PI / 2) * dir), (2 * M_PI));

  if (target_yaw > M_PI)
  {
    target_yaw -= 2 * M_PI;
  }
  if (target_yaw < -M_PI)
  {
    target_yaw += 2 * M_PI;
  }

  printf("Yaw actual: %f, Yaw objetivo: %f\n", c_yaw, target_yaw);

  double turn_speed = MAX_SPEED * TURN_SPEED_FACTOR;
  wb_motor_set_velocity(left_motor, -turn_speed * dir);
  wb_motor_set_velocity(right_motor, turn_speed * dir);

  while (wb_robot_step(TIME_STEP) != -1)
  {
    rpy = wb_inertial_unit_get_roll_pitch_yaw(iu);
    if (fabs(rpy[2] - target_yaw) < 0.03)
    {
      break;
    }
  }
  // Continuar hacia adelante
  wb_motor_set_velocity(left_motor, MAX_SPEED);
  wb_motor_set_velocity(right_motor, MAX_SPEED);
}
```

Mover robot hacia adelante

```
  // Funcion para mover el robot hacia adelante
void forward(WbDeviceTag left_motor, WbDeviceTag right_motor)
{
  wb_motor_set_velocity(left_motor, MAX_SPEED);
  wb_motor_set_velocity(right_motor, MAX_SPEED);
}
```

Función principal

```
int main(int argc, char **argv)
{
  // Inicializando el entorno Webots
  wb_robot_init();

  // Establecer el tiempo de paso para la simulacion
  int time_step = (int)wb_robot_get_basic_time_step();

  // Habilitar y obtener el sensor ultrasonico frontal y configurar su tiempo de
    paso.
  WbDeviceTag front_ultrasonic_sensor = wb_robot_get_device("front ultrasonic sensor
    ");
  wb_distance_sensor_enable(front_ultrasonic_sensor, time_step);
```

```
12  WbDeviceTag right_ultrasonic_sensor = wb_robot_get_device("right ultrasonic sensor
      ");
13  wb_distance_sensor_enable(right_ultrasonic_sensor, time_step);
14
15  // Obtener y configurar los motores izquierdo y derecho
16  WbDeviceTag left_motor = wb_robot_get_device("left wheel motor");
17  WbDeviceTag right_motor = wb_robot_get_device("right wheel motor");
18  wb_motor_set_position(left_motor, INFINITY);
19  wb_motor_set_position(right_motor, INFINITY);
20  wb_motor_set_velocity(left_motor, 0.0); // Establecer velocidad inicial a 0
21  wb_motor_set_velocity(right_motor, 0.0);
22
23  // Unidad inercial
24  WbDeviceTag iu = wb_robot_get_device("inertial unit");
25  wb_inertial_unit_enable(iu, time_step);
26
27  bool flag = true;
28  bool flag_after_right = false;
29
30  char str_message[150];
31
32  while (wb_robot_step(time_step) != -1)
33  {
34    // Leer distancia del sensor ultrasonico
35    float front_distance = wb_distance_sensor_get_value(front_ultrasonic_sensor);
36    float right_distance = wb_distance_sensor_get_value(right_ultrasonic_sensor);
37
38    // Aplicar el filtro de Kalman para el sensor frontal
39    kalman_predict(x_front_k, P_front_k);
40    kalman_update(x_front_k, P_front_k, front_distance);
41    float front_distance_kalman = x_front_k[0];
42
43    // Aplicar el filtro de Kalman para el sensor derecho
44    kalman_predict
45
46        (x_right_k, P_right_k);
47    kalman_update(x_right_k, P_right_k, right_distance);
48    float right_distance_kalman = x_right_k[0];
49
50    // Decidir el movimiento del robot
51    bool there_front_wall = front_distance_kalman <= DISTANCE_THRESHOLD;
52    bool there_right_wall = right_distance_kalman <= DISTANCE_THRESHOLD;
53    bool no_walls = !there_front_wall && !there_right_wall;
54
55    if (no_walls && flag)
56    { // El robot acaba de empezar
57      strcpy(str_message, "Sin paredes");
58      forward(left_motor, right_motor);
59    }
60    else if (there_front_wall)
61    { // Girar a la derecha
62      strcpy(str_message, "Hay una pared enfrente");
63      turn(1, left_motor, right_motor, iu);   // Girar a la izquierda (dir=1)
64      wb_motor_set_velocity(left_motor, 0.0); // Establecer velocidad inicial a 0
65      wb_motor_set_velocity(right_motor, 0.0);
66      wb_robot_step(TIME_STEP * 18);
67      flag = false;
68      flag_after_right = false;
```

```
    }
    else if (there_right_wall)
    {
      strcpy(str_message, "Hay una pared a la derecha");
      forward(left_motor, right_motor);
      flag_after_right = true;
    }
    else if (no_walls && flag_after_right)
    {
      strcpy(str_message, "Sin paredes, girar 90 a la derecha");
      forward(left_motor, right_motor);
      wb_robot_step(TIME_STEP * 18);
      turn(-1, left_motor, right_motor, iu); // Girar a la derecha (dir=-1)
      forward(left_motor, right_motor);
      wb_robot_step(TIME_STEP * 30);
      flag_after_right = false;
      wb_motor_set_velocity(left_motor, 0.0); // Establecer velocidad inicial a 0
      wb_motor_set_velocity(right_motor, 0.0);
      wb_robot_step(TIME_STEP * 18);
    }
    // Imprimir informacion para depuracion
    printf("mensaje: %s\n", str_message);
    printf("distancia frontal (Kalman): %f\n", front_distance_kalman);
    printf("distancia frontal (sensor): %f\n", front_distance);
    printf("distancia derecha (Kalman): %f\n", right_distance_kalman);
    printf("distancia derecha (sensor): %f\n", right_distance);
    printf("-------------------------\n");
  };

  // Limpiar antes de salir
  wb_robot_cleanup();

  return 0;
}
```

6 Conclusiones

El uso del filtro de kalman me ayudo a solucionar el problema con los sensores ultrasónicos, ya que las lecturas que obtenía variaban bastante y si colocaba un umbral de 0.3 metros el robot no lograba establecer correctamente su decisión de giro, en cambio usando el filtro de kalman logre que las mediciones de distancia no variaran demasiado y el robot pudiera actuar correctamente en la toma de decisiones en base al umbral y la distancia obtenida por kalman.

De igual manera, la Implementación del giro de 90 grados fue algo que se me complico hacer, pero una vez obtenidos estas dos piezas clave, el resto solo fue aplicar la lógica para hacer que el robot pudiera completar el laberinto.