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 siempre intenta girar a la derecha si no hay una pared. Si hay una pared frente a él, gira a la derecha. Si no hay pared a la izquierda, gira 270 grados a la izquierda para seguir la pared.

3 Roboto Khepera IV

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

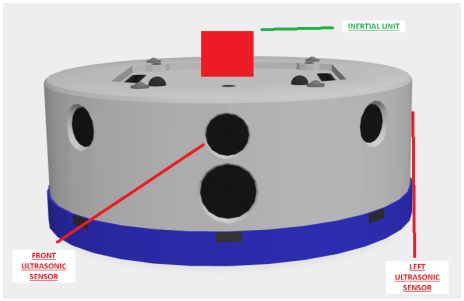

Figure 1: Esquema del robot utilizado

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.

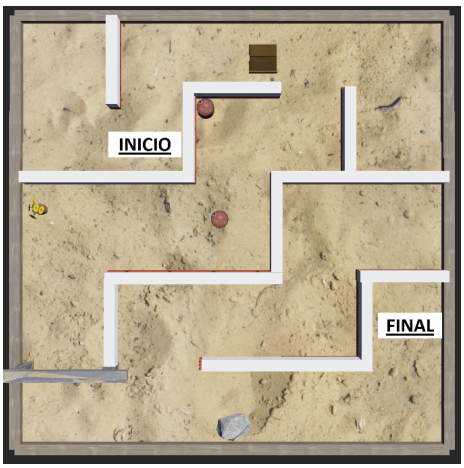

Figure 2: Esquema del laberinto utilizado en las pruebas.

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
```

Declaramos variables globales para el Filtro de Kalman.

```
float x_front_k[2] = {0, 0};
float P_front_k[2][2] = {{1, 0}, {0, 1}};
float F[2][2] = {{1, TIME_STEP / 1000.0}, {0, 1}};
float Q[2][2] = {{1e-4, 0}, {0, 1e-4}};
float H[2] = {1, 0};
float R = 0.05;
float I[2][2] = {{1, 0}, {0, 1}};
float x_left_k[2] = {0, 0};
float P_left_k[2][2] = {{1, 0}, {0, 1}};
```

Función para predecir el estado usando el Filtro de Kalman.

Filtro de kalman, referencia al usuario de github @GogiPuttar (lo tenia en python, yo modifique y adapte al proyecto en lenguaje C).

Estimación del estado predicho

```
void kalman_predict(float x_k[], float P_k[][2]) {
  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];
```

Estimación 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];
    }
}
```

Actualización del estado y la 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];
    }
}
}
```

Función para actualizar el estado usando el Filtro de Kalman.
Innovación o residuo de medición

```
void kalman_update(float x_k[], float P_k[][2], float z) {
  float y_k = z - (H[0] * x_k[0] + H[1] * x_k[1]);
```

Covarianza de la innovación

```
  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;
```

Estimación 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];
    }
  }
```

Actualización de la covarianza

```
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      P_k[i][j] = P_k_1[i][j];
    }
  }
}
```

Función para girar el robot 90 grados en la dirección especificada. referencia al usuario de github @azmonreal (lo tenia en python yo lo modifique y adapte para C y para mi robot)

```
void turn(int dir, WbDeviceTag left_motor, WbDeviceTag right_motor, WbDeviceTag iu)
    {
  double *rpy = wb_inertial_unit_get_roll_pitch_yaw(iu);
  double current_yaw = rpy[2];
```

Calcular el yaw objetivo basado en la dirección de giro

```
  double target_yaw = current_yaw + dir * M_PI / 2;
```

Asegurarse de que el yaw objetivo esté en el rango [-pi, pi]

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

Girar el robot hasta alcanzar el yaw objetivo

```
  while (fabs(current_yaw - target_yaw) > 0.01) {
    wb_motor_set_velocity(left_motor, dir * TURN_SPEED_FACTOR * MAX_SPEED);
    wb_motor_set_velocity(right_motor, -dir * TURN_SPEED_FACTOR * MAX_SPEED);
    wb_robot_step(TIME_STEP);
    rpy = wb_inertial_unit_get_roll_pitch_yaw(iu);
    current_yaw = rpy[2];
  }
```

Detener los motores

```
  wb_motor_set_velocity(left_motor, 0);
  wb_motor_set_velocity(right_motor, 0);
}
```

Función para girar el robot 270 grados a la izquierda (3 veces 90 grados hacia la izquierda).

```
    void turn_270_left(WbDeviceTag left_motor, WbDeviceTag right_motor, WbDeviceTag
    iu) {
      for (int i = 0; i < 3; i++) {
        turn(-1, left_motor, right_motor, iu);
      }
    }
```

Función para mover el robot hacia adelante.

```
void forward(WbDeviceTag left_motor, WbDeviceTag right_motor) {
  wb_motor_set_velocity(left_motor, 0.5 * MAX_SPEED);
  wb_motor_set_velocity(right_motor, 0.5 * MAX_SPEED);
}
```

Función principal donde se inicializan los dispositivos y se ejecuta el comportamiento del robot.

```
int main(int argc, char **argv) {
  wb_robot_init();
  int time_step = (int)wb_robot_get_basic_time_step();
```

Habilitar y obtener el sensor ultrasónico frontal y establecer su tiempo de paso.

```
  WbDeviceTag front_ultrasonic_sensor = wb_robot_get_device("front ultrasonic sensor
    ");
  wb_distance_sensor_enable(front_ultrasonic_sensor, time_step);
  WbDeviceTag left_ultrasonic_sensor = wb_robot_get_device("left ultrasonic sensor")
    ;
  wb_distance_sensor_enable(left_ultrasonic_sensor, time_step);
```

Obtener y configurar los motores izquierdo y derecho

```
WbDeviceTag left_motor = wb_robot_get_device("left wheel motor");
WbDeviceTag right_motor = wb_robot_get_device("right wheel motor");
wb_motor_set_position(left_motor, INFINITY);
wb_motor_set_position(right_motor, INFINITY);
wb_motor_set_velocity(left_motor, 0.0); Establecer la velocidad inicial a 0
wb_motor_set_velocity(right_motor, 0.0);
```

Inertial Unit

```
WbDeviceTag iu = wb_robot_get_device("inertial unit");
wb_inertial_unit_enable(iu, time_step);
```

Banderas y mensaje

```
bool flag = true;
bool flag_after_left = false;
char str_message[150];
```

Leer distancia del sensor ultrasónico

```
while (wb_robot_step(time_step) != -1) {
  float front_distance = wb_distance_sensor_get_value(front_ultrasonic_sensor);
  float left_distance = wb_distance_sensor_get_value(left_ultrasonic_sensor);
```

Aplicar el filtro de Kalman para el sensor frontal

```
  kalman_predict(x_front_k, P_front_k);
  kalman_update(x_front_k, P_front_k, front_distance);
  float front_distance_kalman = x_front_k[0];
```

Aplicar el filtro de Kalman para el sensor izquierdo

```
  kalman_predict(x_left_k, P_left_k);
  kalman_update(x_left_k, P_left_k, left_distance);
  float left_distance_kalman = x_left_k[0];
```

Decidir el movimiento del robot

```
  bool there_front_wall = front_distance_kalman <= DISTANCE_THRESHOLD;
  bool there_left_wall = left_distance_kalman <= DISTANCE_THRESHOLD;
  bool no_walls = !there_front_wall && !there_left_wall;
```

El robot acaba de empezar

```
  if (no_walls && flag) { r
    strcpy(str_message, "No walls");
    forward(left_motor, right_motor);
  }
```

Si no hay pared frontal gira a la derecha 90 grados

```
else if (there_front_wall) {
        strcpy(str_message, "There is a wall in front");
        turn(-1, left_motor, right_motor, iu);
        flag = false;
        flag_after_left = false;
    }
```

Si hay pared izquierda el robot avanza hacia adelante

```
else if (there_left_wall) {
        strcpy(str_message, "There is a wall on the left");
        forward(left_motor, right_motor);
        flag_after_left = true;
    }
```

Si no hay pared frontal ni izquierda, pero antes hubo pared izquierda, el robot gira 270 grados a la derecha.

```
    else if (no_walls && flag_after_left) {
        strcpy(str_message, "No walls and no left wall, turning 270 degrees left");
        forward(left_motor, right_motor);
        wb_robot_step(TIME_STEP * 18);
        turn_270_left(left_motor, right_motor, iu);
        forward(left_motor, right_motor);
        wb_robot_step(TIME_STEP * 32);
        flag_after_left = false;
    }
```

Sino se cumple ningún caso, el robot se detiene

```
    else {
        strcpy(str_message, "no case");
        wb_motor_set_velocity(left_motor, 0.0);
        wb_motor_set_velocity(right_motor, 0.0);
    }
```

Imprimir información para depuración

```
printf("message: %s\n", str_message);
printf("front distance (Kalman): %f\n", front_distance_kalman);
printf("front distance (sensor): %f\n", front_distance);
printf("left distance (Kalman): %f\n", left_distance_kalman);
printf("left distance (sensor): %f\n", left_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.