SIMULATION WEBOT ROS2

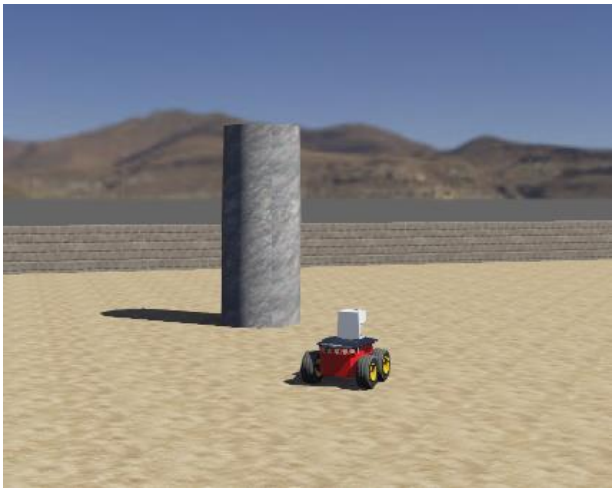
Oleh:

Thasya Mulia (1103201208)

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO UNIVERSITAS TELKOM

2023



Pemaparan

Gambar di atas merupakan sebuah tangkapan layer dari hasil simulasi kontrol mobil otonom yang menggunakan sensor LIDAR pada Webots. Sensor LIDAR digunakan untuk mengukur jarak antara mobil dengan objek sekitarnya. LIDAR (Light Detection and Ranging) adalah sebuah sensor yang menggunakan sinar laser untuk mengukur jarak antara objek atau permukaan tertentu dan perangkat pemindai. Prinsip kerja LIDAR adalah dengan memancarkan sinar laser ke objek dan kemudian mengukur waktu yang dibutuhkan untuk pantulan sinar laser kembali ke perangkat pemindai, sehingga dapat mengetahui jarak dari objek tersebut. Data yang dihasilkan oleh LIDAR dapat digunakan untuk berbagai aplikasi, seperti dalam industri otomotif untuk sistem kendali jarak dekat dan kendaraan tanpa pengemudi, serta di bidang penginderaan jauh untuk pemetaan, pemantauan lingkungan, dan sebagainya.

Simulasi ini menggunakan konsep Braitenberg Vehicle untuk mengontrol kecepatan mobil. Konsep Braitenberg Vehicle adalah sebuah model sederhana dalam robotika yang dikembangkan oleh ilmuwan neurobiologi Valentino Braitenberg pada tahun 1984. Model ini didasarkan pada prinsip-prinsip sederhana dari fisika dan neurobiologi untuk memahami perilaku dasar pada robot.

Braitenberg Vehicle menggambarkan sebuah robot yang terdiri dari sensor dan motor. Sensor dapat mendeteksi cahaya, suara, atau objek lain di sekitarnya, sedangkan motor menggerakkan robot tersebut. Dalam konsep ini, Braitenberg menggambarkan empat tipe dasar dari robot:

Tipe 1: Robot dengan sensor yang langsung terhubung ke motor, sehingga gerakan robot akan langsung dipengaruhi oleh input sensor.

Tipe 2: Robot dengan sensor yang terhubung dengan motor melalui sinyal yang diinverskan, sehingga gerakan robot akan berbeda dengan input sensor.

Tipe 3: Robot dengan sensor yang terhubung dengan motor melalui sinyal yang diinverskan dan dikuadratkan, sehingga gerakan robot menjadi lebih sensitif terhadap input sensor.

Tipe 4: Robot dengan dua sensor dan dua motor, sehingga gerakan robot dipengaruhi oleh interaksi antara kedua sensor.

Penjelasan bagian program :

```
#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <webots/lidar.h>

#include <webots/motor.h>

#include <webots/robot.h>
```

Bagian ini adalah bagian deklarasi header file yang diperlukan oleh program. Header file yang digunakan antara lain adalah math.h untuk fungsi matematika, stdio.h untuk fungsi input-output standar, stdlib.h untuk alokasi memori dinamis, webots/lidar.h dan webots/motor.h untuk penggunaan perangkat dan webots/robot.h untuk inisialisasi robot.

```
#define TIME_STEP 32

#define MAX_SPEED 6.4

#define CRUISING_SPEED 5.0

#define OBSTACLE_THRESHOLD 0.1

#define DECREASE_FACTOR 0.9

#define BACK_SLOWDOWN 0.9
```

Bagian ini adalah definisi konstanta yang digunakan dalam program. Konstanta-konstanta tersebut adalah:

- TIME_STEP adalah interval waktu dalam milidetik pada setiap siklus kontrol.
- MAX_SPEED adalah kecepatan maksimum yang dapat dicapai oleh roda belakang robot.
- CRUISING_SPEED adalah kecepatan roda saat robot sedang melaju tanpa ada halangan.
- OBSTACLE_THRESHOLD adalah batas ambang untuk menentukan apakah ada halangan atau tidak.
- DECREASE_FACTOR adalah faktor pengurangan kecepatan saat ada halangan.
- BACK_SLOWDOWN adalah faktor perlambatan pada roda belakang saat robot harus mundur.

```
// fungsi gaussian

double gaussian(double x, double mu, double sigma) {

  return (1.0 / (sigma * sqrt(2.0 * M_PI))) * exp(-((x - mu) * (x - mu)) / (2 * sigma * sigma));

}
```

Bagian ini adalah fungsi matematika untuk menghitung nilai Gaussian.

```
int main(int argc, char **argv) {
  // inisialisasi perangkat Webots
  wb_robot_init();
```

Bagian ini adalah fungsi utama main pada program yang berisi inisialisasi awal robot dan perangkat yang akan digunakan.

```
  // mendapatkan perangkat
  WbDeviceTag lms291 = wb_robot_get_device("Sick LMS 291");
  WbDeviceTag front_left_wheel = wb_robot_get_device("front left wheel");
  WbDeviceTag front_right_wheel = wb_robot_get_device("front right wheel");
  WbDeviceTag back_left_wheel = wb_robot_get_device("back left wheel");
  WbDeviceTag back_right_wheel = wb_robot_get_device("back right wheel");
```

Bagian ini adalah inisialisasi perangkat yang akan digunakan pada robot. lms291 adalah perangkat sensor LIDAR, sedangkan front_left_wheel, front_right_wheel, back_left_wheel, dan back_right_wheel adalah perangkat motor untuk menggerakkan roda.

```
  // inisialisasi lms291
  wb_lidar_enable(lms291, TIME_STEP);
  const int lms291_width = wb_lidar_get_horizontal_resolution(lms291);
  const int half_width = lms291_width / 2;
  const int max_range = wb_lidar_get_max_range(lms291);
  const double range_threshold = max_range / 20.0;
  const float *lms291_values = NULL;
```

Bagian ini melakukan inisialisasi terhadap sensor LIDAR dan menentukan beberapa parameter seperti lebar resolusi horizontal, setengah lebar resolusi, jarak maksimum yang dapat diukur oleh LIDAR, dan ambang batas jarak untuk mengidentifikasi apakah suatu rintangan terdeteksi atau tidak.

```
  // inisialisasi koefisien braitenberg
  double      *const      braitenberg_coefficients      =      (double
*)malloc(sizeof(double) * lms291_width);
  int i, j;
  for (i = 0; i < lms291_width; ++i)
    braitenberg_coefficients[i] = gaussian(i, half_width, lms291_width /
5);
```

Bagian ini melakukan inisialisasi koefisien Braitenberg yang akan digunakan untuk mengubah data LIDAR menjadi informasi tentang rintangan di sekitar robot. Setiap elemen koefisien Braitenberg adalah nilai Gaussian yang menggambarkan bobot kepentingan suatu data LIDAR dalam mengidentifikasi rintangan.

```
  // inisialisasi motor
  wb_motor_set_position(front_left_wheel, INFINITY);
  wb_motor_set_position(front_right_wheel, INFINITY);
  wb_motor_set_position(back_left_wheel, INFINITY);
  wb_motor_set_position(back_right_wheel, INFINITY);

  // inisialisasi kecepatan untuk setiap roda
  double back_left_speed = 0.0, back_right_speed = 0.0;
  double front_left_speed = 0.0, front_right_speed = 0.0;
  wb_motor_set_velocity(front_left_wheel, front_left_speed);
  wb_motor_set_velocity(front_right_wheel, front_right_speed);
  wb_motor_set_velocity(back_left_wheel, back_left_speed);
  wb_motor_set_velocity(back_right_wheel, back_right_speed);
```

Bagian ini melakukan inisialisasi terhadap motor yang digunakan untuk menggerakkan robot. Setiap motor diatur ke posisi tak terhingga dan kecepatan awal diatur ke nol.

```
  // inisialisasi variabel dinamis
  double left_obstacle = 0.0, right_obstacle = 0.0;
```

Bagian ini melakukan inisialisasi terhadap variabel dinamis yang akan digunakan untuk menyimpan informasi tentang rintangan di sekitar robot.

```
// loop kontrol
  while (wb_robot_step(TIME_STEP) != -1) {
```

Bagian ini merupakan loop utama yang mengontrol perilaku robot selama program berjalan.

```
 // dapatkan nilai lidar
    lms291_values = wb_lidar_get_range_image(lms291);
```

Bagian ini mengambil data LIDAR dari sensor LIDAR.

```
// terapkan koefisien braitenberg pada nilai yang dihasilkan dari lms291
    // hambatan dekat yang terdeteksi di sisi kiri
    for (i = 0; i < half_width; ++i) {
      if (lms291_values[i] < range_threshold)  // hambatan jauh diabaikan
        left_obstacle   +=   braitenberg_coefficients[i]   *   (1.0   -
lms291_values[i] / max_range);
      // hambatan dekat yang terdeteksi di sisi kanan
      j = lms291_width - i - 1;
      if (lms291_values[j] < range_threshold)
        right_obstacle   +=   braitenberg_coefficients[i]   *   (1.0   -
lms291_values[j] / max_range);
    }
```

Melakukan iterasi pada setengah lebar LMS291, dimana setiap nilai yang diterima oleh sensor LMS291 akan diperiksa apakah nilai tersebut lebih kecil dari threshold jarak atau tidak. Jika nilai tersebut lebih kecil dari threshold jarak, maka nilai left_obstacle dan right_obstacle akan dihitung dengan menggunakan koefisien Braitenberg yang sudah diatur dan disesuaikan dengan jarak antara robot dengan objek.

```
// total hambatan depan
    const double obstacle = left_obstacle + right_obstacle;
```

Menghitung total nilai left_obstacle dan right_obstacle sebagai obstacle keseluruhan.

```
 // hitung kecepatan sesuai dengan informasi tentang
    // hambatan
    if (obstacle > OBSTACLE_THRESHOLD) {
      const double speed_factor = (1.0 - DECREASE_FACTOR * obstacle) *
MAX_SPEED / obstacle;
      front_left_speed = speed_factor * left_obstacle;
      front_right_speed = speed_factor * right_obstacle;
      back_left_speed = BACK_SLOWDOWN * front_left_speed;
      back_right_speed = BACK_SLOWDOWN * front_right_speed;
```

Jika nilai obstacle lebih besar dari OBSTACLE_THRESHOLD, maka robot akan melambatkan kecepatannya dan mengubah kecepatan roda pada sisi kiri dan kanan. Perubahan kecepatan roda tersebut diatur berdasarkan faktor kecepatan dan obstacle yang ada.

```
   } else {
      back_left_speed = CRUISING_SPEED;
      back_right_speed = CRUISING_SPEED;
      front_left_speed = CRUISING_SPEED;
      front_right_speed = CRUISING_SPEED;
    }
```

Jika nilai obstacle lebih kecil dari OBSTACLE_THRESHOLD, maka kecepatan roda akan dikembalikan ke kecepatan default (CRUISING_SPEED).

```
  // set actuator
    wb_motor_set_velocity(front_left_wheel, front_left_speed);
    wb_motor_set_velocity(front_right_wheel, front_right_speed);
    wb_motor_set_velocity(back_left_wheel, back_left_speed);
    wb_motor_set_velocity(back_right_wheel, back_right_speed);
```

Mengatur kecepatan putaran roda dengan menggunakan fungsi wb_motor_set_velocity pada setiap roda.

```
    // reset variabel dinamis menjadi nol
    left_obstacle = 0.0;
    right_obstacle = 0.0;
  }
```

Menetapkan nilai left_obstacle dan right_obstacle kembali ke 0 setelah nilai tersebut digunakan dalam penghitungan kecepatan roda.

```
 free(braitenberg_coefficients);
```

Membebaskan memori yang digunakan oleh braitenberg_coefficients.

```
wb_robot_cleanup();
```

Membersihkan penggunaan robot pada simulasi.

```
  return 0;
}
```

Mengembalikan nilai 0.