

Robotics Technical Report

“Hacking Robotic Programming Using Mavic demo robot ”



Universitas Telkom

Oleh:

Muhammad Haidar Abdul Jabbar

1103202071

Program Studi Teknik Komputer

Fakultas Teknik Elektro

Universitas Telkom

2022/2023



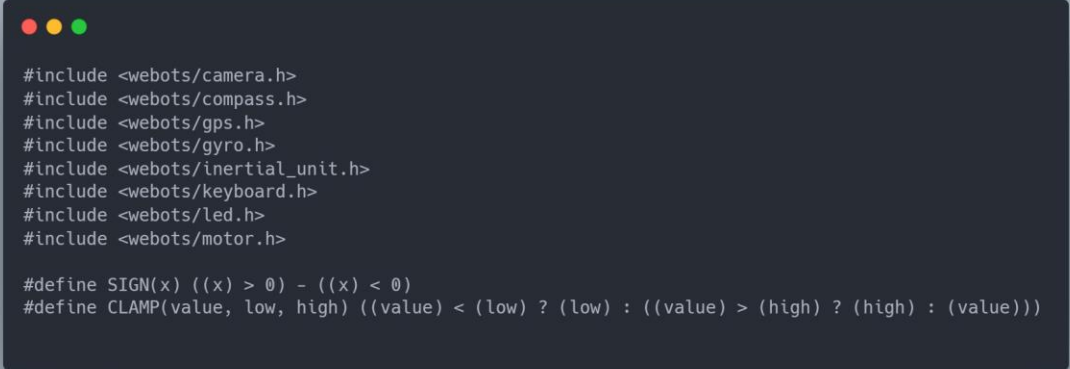
Mavic Robot dalam Webots adalah model robot simulasi yang didasarkan pada produk Mavic Mini, sebuah drone yang diproduksi oleh DJI. Webots sendiri adalah platform simulasi robot yang digunakan untuk mengembangkan dan menguji algoritma dan kontroler robot sebelum diimplementasikan pada robot fisik.

Mavic Robot dalam Webots mencoba mereplikasi perilaku dan fitur dari drone Mavic Mini dalam lingkungan simulasi. Ini memungkinkan pengembang untuk menguji dan memvalidasi algoritma navigasi, penginderaan, pengendalian, dan interaksi dengan lingkungan sebelum menerapkannya pada drone fisik.

Dalam simulasi Webots, Mavic Robot dapat dikendalikan menggunakan kode pemrograman yang memanipulasi input dan output dari sensor dan aktuator yang ada pada drone. Pengguna dapat mengontrol gerakan, menerapkan logika navigasi, mengumpulkan data dari sensor seperti kamera atau sensor jarak, dan merespons lingkungan virtual di sekitarnya.

Penggunaan Mavic Robot dalam Webots memberikan kesempatan untuk menguji dan mengembangkan algoritma dan kontroler dengan biaya rendah dan risiko minimal sebelum mengimplementasikannya pada drone fisik. Ini mempercepat dan mempermudah proses pengembangan, serta membantu dalam mengurangi kesalahan atau risiko yang mungkin terjadi pada tahap awal pengembangan.

Penjelasan Source Code



```
#include <webots/camera.h>
#include <webots/compass.h>
#include <webots/gps.h>
#include <webots/gyro.h>
#include <webots/inertial_unit.h>
#include <webots/keyboard.h>
#include <webots/led.h>
#include <webots/motor.h>

#define SIGN(x) ((x) > 0) - ((x) < 0)
#define CLAMP(value, low, high) ((value) < (low) ? (low) : ((value) > (high) ? (high) : (value)))
```

#include <math.h>:

Mengimport header file math.h, yang berisi fungsi-fungsi matematika dasar seperti fungsi trigonometri, fungsi eksponensial, dll.

#include <stdio.h>:

Mengimport header file stdio.h, yang berisi fungsi-fungsi standar untuk input/output dalam bahasa C, seperti fungsi printf() dan scanf().

#include <stdlib.h>:

Mengimport header file stdlib.h, yang berisi fungsi-fungsi standar untuk pengaturan memori, pengolahan string, konversi tipe data, dan fungsi-fungsi utilitas lainnya seperti malloc() dan exit().

#include <webots/robot.h>:

Mengimport header file robot.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan kontrol robot dalam simulasi Webots.

#include <webots/camera.h>:

Mengimport header file camera.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan kamera dalam simulasi Webots.

#include <webots/compass.h>:

Mengimport header file compass.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan kompas dalam simulasi Webots.

#include <webots/gps.h>:

Mengimport header file gps.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan GPS (Global Positioning System) dalam simulasi Webots.

#include <webots/gyro.h>:

Mengimport header file gyro.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan sensor gyro dalam simulasi Webots.

#include <webots/inertial_unit.h>:

Mengimport header file inertial_unit.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan unit inersia (IMU) dalam simulasi Webots.

#include <webots/keyboard.h>:

Mengimport header file keyboard.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan interaksi dengan keyboard dalam simulasi Webots.

#include <webots/led.h>:

Mengimport header file led.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan LED dalam simulasi Webots.

#include <webots/motor.h>:

Mengimport header file motor.h dari Webots, yang berisi definisi dan fungsi-fungsi yang berkaitan dengan penggunaan motor dalam simulasi Webots.

#define SIGN(x) ((x) > 0) - ((x) < 0)

Makro SIGN digunakan untuk mengembalikan tanda suatu nilai numerik x, mirip dengan penjelasan sebelumnya. Jika x positif, maka ekspresi $((x) > 0)$ menghasilkan nilai 1. Jika x negatif, maka ekspresi $((x) < 0)$ menghasilkan nilai 1. Jika x nol, maka kedua ekspresi menghasilkan nilai 0. Oleh karena itu, makro SIGN akan mengembalikan nilai 1 untuk nilai positif, -1 untuk nilai negatif, dan 0 untuk nilai nol.

#define CLAMP(value, low, high) ((value) < (low) ? (low) : ((value) > (high) ? (high) : (value)))

Makro CLAMP juga digunakan untuk membatasi suatu nilai value agar berada di antara low dan high, mirip dengan penjelasan sebelumnya. Jika value lebih kecil dari low, maka makro akan mengembalikan nilai low. Jika value lebih besar dari high, maka makro akan mengembalikan nilai high. Jika value berada di antara low dan high, maka makro akan mengembalikan value itu sendiri. Dengan demikian, makro CLAMP memastikan bahwa value tidak melampaui batasan low dan high.

```

wb_robot_init();
int timestep = (int)wb_robot_get_basic_time_step();

// Get and enable devices.
WbDeviceTag camera = wb_robot_get_device("camera");
wb_camera_enable(camera, timestep);
WbDeviceTag front_left_led = wb_robot_get_device("front left led");
WbDeviceTag front_right_led = wb_robot_get_device("front right led");
WbDeviceTag imu = wb_robot_get_device("inertial unit");
wb_inertial_unit_enable(imu, timestep);
WbDeviceTag gps = wb_robot_get_device("gps");
wb_gps_enable(gps, timestep);
WbDeviceTag compass = wb_robot_get_device("compass");
wb_compass_enable(compass, timestep);
WbDeviceTag gyro = wb_robot_get_device("gyro");
wb_gyro_enable(gyro, timestep);
wb_keyboard_enable(timestep);
WbDeviceTag camera_roll_motor = wb_robot_get_device("camera roll");
WbDeviceTag camera_pitch_motor = wb_robot_get_device("camera pitch");

```

Kode yang diberikan menginisialisasi simulasi robot dalam Webots dan mengaktifkan beberapa perangkat (devices) yang akan digunakan dalam simulasi. Penjelasan baris kode:

wb_robot_init();

Fungsi `wb_robot_init()` digunakan untuk menginisialisasi robot dalam simulasi Webots. Ini harus dipanggil sebelum menggunakan fungsi-fungsi lain dalam Webots.

int timestep = (int)wb_robot_get_basic_time_step();

Baris ini mengambil nilai waktu dasar (basic time step) dari simulasi menggunakan fungsi `wb_robot_get_basic_time_step()`, dan menyimpannya dalam variabel `timestep`. Nilai ini akan digunakan untuk mengatur kecepatan pembaruan simulasi.

WbDeviceTag camera = wb_robot_get_device("camera");

Baris ini mendapatkan tag perangkat (device tag) untuk kamera dalam simulasi dengan nama "camera". `WbDeviceTag` adalah tipe data yang merepresentasikan perangkat dalam Webots.

wb_camera_enable(camera, timestep);

Baris ini mengaktifkan kamera yang telah didapatkan sebelumnya dengan menggunakan fungsi `wb_camera_enable()`. Argumen `camera` adalah tag perangkat kamera, dan `timestep` adalah waktu dasar yang digunakan untuk pembaruan kamera.

WbDeviceTag front_left_led = wb_robot_get_device("front left led");

Baris ini mendapatkan tag perangkat untuk LED di bagian depan sebelah kiri robot dengan nama "front left led".

WbDeviceTag front_right_led = wb_robot_get_device("front right led");

Baris ini mendapatkan tag perangkat untuk LED di bagian depan sebelah kanan robot dengan nama "front right led".

WbDeviceTag imu = wb_robot_get_device("inertial unit");

Baris ini mendapatkan tag perangkat untuk unit inersia (inertial unit) dalam robot dengan nama "inertial unit".

wb_inertial_unit_enable(imu, timestep);

Baris ini mengaktifkan unit inersia yang telah didapatkan sebelumnya dengan menggunakan fungsi `wb_inertial_unit_enable()`. Argumen `imu` adalah tag perangkat unit inersia, dan `timestep` adalah waktu dasar yang digunakan untuk pembaruan unit inersia.

WbDeviceTag gps = wb_robot_get_device("gps");

Baris ini mendapatkan tag perangkat untuk GPS dalam simulasi dengan nama "gps".

wb_gps_enable(gps, timestep);

Baris ini mengaktifkan GPS yang telah didapatkan sebelumnya dengan menggunakan fungsi `wb_gps_enable()`. Argumen `gps` adalah tag perangkat GPS, dan `timestep` adalah waktu dasar yang digunakan untuk pembaruan GPS.

WbDeviceTag compass = wb_robot_get_device("compass");

Baris ini mendapatkan tag perangkat untuk kompas dalam simulasi dengan nama "compass".

wb_compass_enable(compass, timestep);

Baris ini mengaktifkan kompas yang telah didapatkan sebelumnya dengan menggunakan fungsi `wb_compass_enable()`. Argumen `compass` adalah tag perangkat kompas, dan `timestep` adalah waktu dasar yang digunakan untuk pembaruan kompas.

WbDeviceTag gyro = wb_robot_get_device("gyro");

Baris ini mendapatkan tag perangkat untuk gyro dalam simulasi dengan nama "gyro".

wb_gyro_enable(gyro, timestep);

Baris ini mengaktifkan gyro yang telah didapatkan sebelumnya dengan menggunakan fungsi `wb_gyro_enable()`. Argumen `gyro` adalah tag perangkat gyro, dan `timestep` adalah waktu dasar yang digunakan untuk pembaruan gyro.

wb_keyboard_enable(timestep);

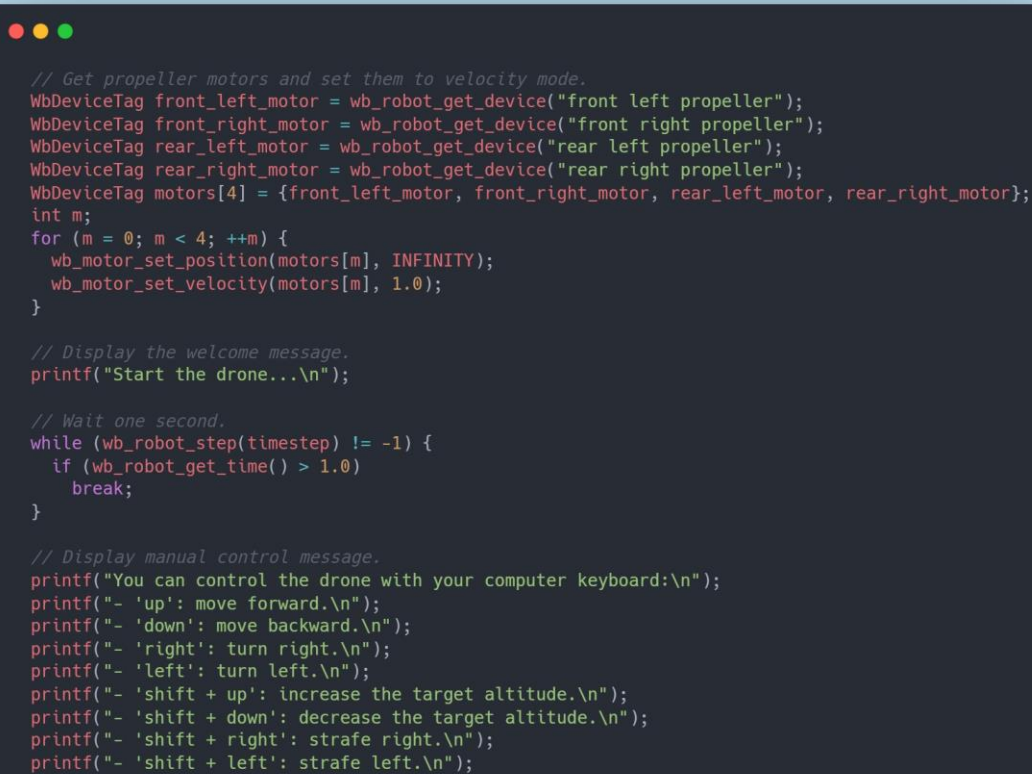
Baris ini mengaktifkan keyboard dalam simulasi dengan menggunakan fungsi `wb_keyboard_enable()`. Argumen `timestep` adalah waktu dasar yang digunakan untuk pembaruan input keyboard.

WbDeviceTag camera_roll_motor = wb_robot_get_device("camera roll");

Baris ini mendapatkan tag perangkat untuk motor roll kamera dalam simulasi dengan nama "camera roll".

WbDeviceTag camera_pitch_motor = wb_robot_get_device("camera pitch");

Baris ini mendapatkan tag perangkat untuk motor pitch kamera dalam simulasi dengan nama "camera pitch".

A screenshot of a code editor window with a dark background and light-colored text. The code is in C++ and is part of a drone simulation. It includes comments in green and code in red. The code sets up four propeller motors, sets their positions and velocities, displays a welcome message, waits for one second, and then displays a manual control message with various keyboard shortcuts.

```
// Get propeller motors and set them to velocity mode.
WbDeviceTag front_left_motor = wb_robot_get_device("front left propeller");
WbDeviceTag front_right_motor = wb_robot_get_device("front right propeller");
WbDeviceTag rear_left_motor = wb_robot_get_device("rear left propeller");
WbDeviceTag rear_right_motor = wb_robot_get_device("rear right propeller");
WbDeviceTag motors[4] = {front_left_motor, front_right_motor, rear_left_motor, rear_right_motor};
int m;
for (m = 0; m < 4; ++m) {
    wb_motor_set_position(motors[m], INFINITY);
    wb_motor_set_velocity(motors[m], 1.0);
}

// Display the welcome message.
printf("Start the drone...\n");

// Wait one second.
while (wb_robot_step(timestep) != -1) {
    if (wb_robot_get_time() > 1.0)
        break;
}

// Display manual control message.
printf("You can control the drone with your computer keyboard:\n");
printf("- 'up': move forward.\n");
printf("- 'down': move backward.\n");
printf("- 'right': turn right.\n");
printf("- 'left': turn left.\n");
printf("- 'shift + up': increase the target altitude.\n");
printf("- 'shift + down': decrease the target altitude.\n");
printf("- 'shift + right': strafe right.\n");
printf("- 'shift + left': strafe left.\n");
```

Kode yang diberikan mengatur motor-motor propeller pada drone dalam simulasi Webots dan menampilkan pesan selamat datang serta pesan kontrol manual pada program. Penjelasan kode:

WbDeviceTag front_left_motor = wb_robot_get_device("front left propeller");

Baris ini mendapatkan tag perangkat untuk motor propeller di bagian depan sebelah kiri drone dengan nama "front left propeller".

WbDeviceTag front_right_motor = wb_robot_get_device("front right propeller");

Baris ini mendapatkan tag perangkat untuk motor propeller di bagian depan sebelah kanan drone dengan nama "front right propeller".

WbDeviceTag rear_left_motor = wb_robot_get_device("rear left propeller");

Baris ini mendapatkan tag perangkat untuk motor propeller di bagian belakang sebelah kiri drone dengan nama "rear left propeller".

WbDeviceTag rear_right_motor = wb_robot_get_device("rear right propeller");

Baris ini mendapatkan tag perangkat untuk motor propeller di bagian belakang sebelah kanan drone dengan nama "rear right propeller".

```
WbDeviceTag motors[4] = {front_left_motor, front_right_motor, rear_left_motor,  
rear_right_motor};
```

Baris ini mendeklarasikan sebuah array motors yang berisi tag perangkat dari keempat motor propeller drone.

```
for (m = 0; m < 4; ++m) { wb_motor_set_position(motors[m], INFINITY);  
wb_motor_set_velocity(motors[m], 1.0); }
```

Dalam loop ini, setiap motor propeller pada drone diatur ke mode kecepatan (velocity mode) dengan menggunakan fungsi `wb_motor_set_position()` dan `wb_motor_set_velocity()`. Parameter `INFINITY` pada `wb_motor_set_position()` mengatur motor ke mode velocity dan memungkinkan pengaturan kecepatan propeller. Di sini, setiap motor diberi kecepatan awal 1.0.

```
printf("Start the drone...\n");
```

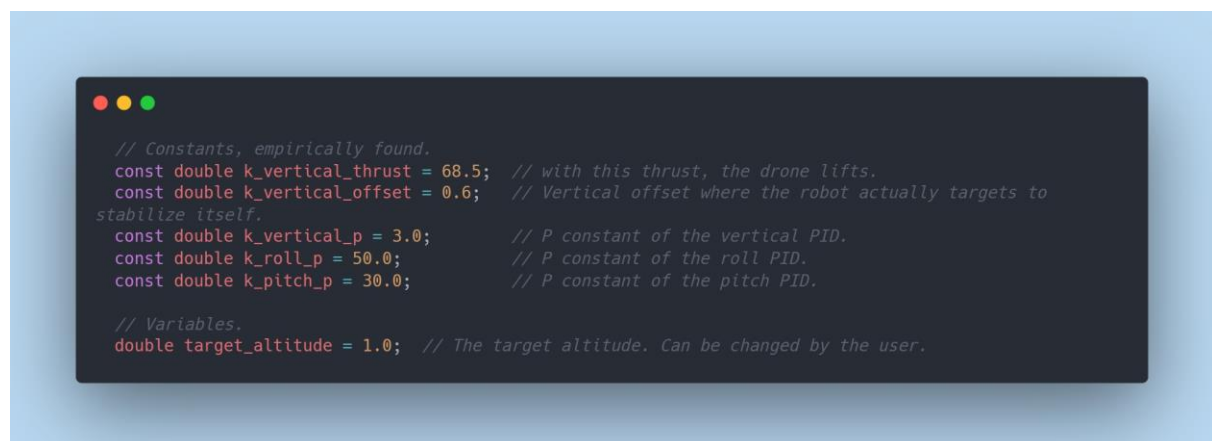
Baris ini mencetak pesan "Start the drone..." ke konsol.

```
while (wb_robot_step(timestep) != -1) { if (wb_robot_get_time() > 1.0) break; }
```

Loop ini memungkinkan program menunggu selama satu detik sebelum melanjutkan ke baris berikutnya. Fungsi `wb_robot_step()` digunakan untuk mengupdate simulasi Webots pada setiap iterasi loop. Loop ini akan berhenti jika waktu simulasi `wb_robot_get_time()` lebih dari 1.0 detik.

```
//Printf
```

Pesan-pesan berikutnya menggunakan `printf()` digunakan untuk menampilkan instruksi pengendalian manual drone kepada pengguna. Instruksi ini mencakup pergerakan maju, mundur, belok kanan, belok kiri, perubahan ketinggian, dan pergerakan lateral menggunakan kombinasi tombol keyboard.



Kode yang diberikan menginisialisasi konstanta dan variabel yang digunakan dalam pengendalian drone. penjelasan kode:

```
const double k_vertical_thrust = 68.5;
```

Baris ini mendefinisikan konstanta `k_vertical_thrust` dengan nilai 68.5. Konstanta ini digunakan untuk menentukan dorongan vertikal yang diperlukan agar drone dapat terbang.

const double k_vertical_offset = 0.6;

Baris ini mendefinisikan konstanta k_vertical_offset dengan nilai 0.6. Konstanta ini mengacu pada offset vertikal dimana drone sebenarnya menargetkan untuk menstabilkan dirinya sendiri.

const double k_vertical_p = 3.0;

Baris ini mendefinisikan konstanta k_vertical_p dengan nilai 3.0. Konstanta ini merupakan konstanta proporsional (P) dari pengendali PID (Proportional-Integral-Derivative) pada sumbu vertikal. Konstanta ini mempengaruhi sejauh mana drone akan bereaksi terhadap perubahan ketinggian yang diinginkan.

const double k_roll_p = 50.0;

Baris ini mendefinisikan konstanta k_roll_p dengan nilai 50.0. Konstanta ini merupakan konstanta proporsional (P) dari pengendali PID pada sumbu roll. Konstanta ini mempengaruhi sejauh mana drone akan bereaksi terhadap perubahan roll yang diinginkan.

const double k_pitch_p = 30.0;

Baris ini mendefinisikan konstanta k_pitch_p dengan nilai 30.0. Konstanta ini merupakan konstanta proporsional (P) dari pengendali PID pada sumbu pitch. Konstanta ini mempengaruhi sejauh mana drone akan bereaksi terhadap perubahan pitch yang diinginkan.

double target_altitude = 1.0;

Baris ini mendeklarasikan variabel target_altitude dengan nilai awal 1.0. Variabel ini merepresentasikan ketinggian target yang diinginkan oleh drone. Nilai ini dapat diubah oleh pengguna sesuai kebutuhan.

```

// Main loop
while (wb_robot_step(timestep) != -1) {
    const double time = wb_robot_get_time(); // in seconds.

    // Retrieve robot position using the sensors.
    const double roll = wb_inertial_unit_get_roll_pitch_yaw(imu)[0];
    const double pitch = wb_inertial_unit_get_roll_pitch_yaw(imu)[1];
    const double altitude = wb_gps_get_values(gps)[2];
    const double roll_acceleration = wb_gyro_get_values(gyro)[0];
    const double pitch_acceleration = wb_gyro_get_values(gyro)[1];

    // Blink the front LEDs alternatively with a 1 second rate.
    const bool led_state = ((int)time) % 2;
    wb_led_set(front_left_led, led_state);
    wb_led_set(front_right_led, !led_state);

    // Stabilize the Camera by actuating the camera motors according to the gyro feedback.
    wb_motor_set_position(camera_roll_motor, -0.115 * roll_acceleration);
    wb_motor_set_position(camera_pitch_motor, -0.1 * pitch_acceleration);

    // Transform the keyboard input to disturbances on the stabilization algorithm.
    double roll_disturbance = 0.0;
    double pitch_disturbance = 0.0;
    double yaw_disturbance = 0.0;
    int key = wb_keyboard_get_key();
    while (key > 0) {
        switch (key) {
            case WB_KEYBOARD_UP:
                pitch_disturbance = -2.0;
                break;
            case WB_KEYBOARD_DOWN:
                pitch_disturbance = 2.0;
                break;
            case WB_KEYBOARD_RIGHT:
                yaw_disturbance = -1.3;
                break;
            case WB_KEYBOARD_LEFT:
                yaw_disturbance = 1.3;
                break;
            case (WB_KEYBOARD_SHIFT + WB_KEYBOARD_RIGHT):
                roll_disturbance = -1.0;
                break;
            case (WB_KEYBOARD_SHIFT + WB_KEYBOARD_LEFT):
                roll_disturbance = 1.0;
                break;
            case (WB_KEYBOARD_SHIFT + WB_KEYBOARD_UP):
                target_altitude += 0.05;
                printf("target altitude: %f [m]\n", target_altitude);
                break;
            case (WB_KEYBOARD_SHIFT + WB_KEYBOARD_DOWN):
                target_altitude -= 0.05;
                printf("target altitude: %f [m]\n", target_altitude);
                break;
        }
        key = wb_keyboard_get_key();
    }

    // Compute the roll, pitch, yaw and vertical inputs.
    const double roll_input = k_roll_p * CLAMP(roll, -1.0, 1.0) + roll_acceleration + roll_disturbance;
    const double pitch_input = k_pitch_p * CLAMP(pitch, -1.0, 1.0) + pitch_acceleration +
    pitch_disturbance;
    const double yaw_input = yaw_disturbance;
    const double clamped_difference_altitude = CLAMP(target_altitude - altitude + k_vertical_offset,
    -1.0, 1.0);
    const double vertical_input = k_vertical_p * pow(clamped_difference_altitude, 3.0);

    // Actuate the motors taking into consideration all the computed inputs.
    const double front_left_motor_input = k_vertical_thrust + vertical_input - roll_input + pitch_input
    - yaw_input;
    const double front_right_motor_input = k_vertical_thrust + vertical_input + roll_input +
    pitch_input + yaw_input;
    const double rear_left_motor_input = k_vertical_thrust + vertical_input - roll_input - pitch_input
    + yaw_input;
    const double rear_right_motor_input = k_vertical_thrust + vertical_input + roll_input - pitch_input
    - yaw_input;
    wb_motor_set_velocity(front_left_motor, front_left_motor_input);
    wb_motor_set_velocity(front_right_motor, -front_right_motor_input);
    wb_motor_set_velocity(rear_left_motor, -rear_left_motor_input);
    wb_motor_set_velocity(rear_right_motor, rear_right_motor_input);
};

```

Kode diatas merupakan bagian dari loop utama program.penjelasan kode:

while (wb_robot_step(timestep) != -1) {

Ini adalah awal dari loop utama. Loop ini akan terus berjalan selama simulasi berjalan dan tidak ada kesalahan.

const double time = wb_robot_get_time();

Baris ini mengambil waktu saat ini dalam detik menggunakan fungsi wb_robot_get_time().

const double roll = wb_inertial_unit_get_roll_pitch_yaw(imu)[0];

Baris ini mendapatkan nilai roll (kecondongan lateral) dari unit inersia menggunakan fungsi `wb_inertial_unit_get_roll_pitch_yaw(imu)[0]`.

```
const double pitch = wb_inertial_unit_get_roll_pitch_yaw(imu)[1];
```

Baris ini mendapatkan nilai pitch (kecondongan longitudinal) dari unit inersia menggunakan fungsi `wb_inertial_unit_get_roll_pitch_yaw(imu)[1]`.

```
const double altitude = wb_gps_get_values(gps)[2];
```

Baris ini mendapatkan nilai ketinggian dari GPS menggunakan fungsi `wb_gps_get_values(gps)[2]`.

```
const double roll_acceleration = wb_gyro_get_values(gyro)[0];
```

Baris ini mendapatkan nilai percepatan roll dari gyro menggunakan fungsi `wb_gyro_get_values(gyro)[0]`.

```
const double pitch_acceleration = wb_gyro_get_values(gyro)[1];
```

Baris ini mendapatkan nilai percepatan pitch dari gyro menggunakan fungsi `wb_gyro_get_values(gyro)[1]`.

```
const bool led_state = ((int)time) % 2;
```

Baris ini menghitung keadaan LED dengan membagi waktu dengan 2 dan mengambil sisa hasil bagi. Jika sisa hasil bagi adalah 0, maka `led_state` bernilai true, jika bukan, maka `led_state` bernilai false.

```
wb_led_set(front_left_led, led_state);
```

Baris ini mengatur LED depan kiri dengan menggunakan fungsi `wb_led_set()` untuk menyalakan atau mematikan LED berdasarkan nilai `led_state`.

```
wb_led_set(front_right_led, !led_state);
```

Baris ini mengatur LED depan kanan dengan menggunakan fungsi `wb_led_set()` untuk menyalakan atau mematikan LED berdasarkan nilai kebalikan dari `led_state`.

```
wb_motor_set_position(camera_roll_motor, -0.115 * roll_acceleration);
```

Baris ini mengatur posisi motor kamera roll berdasarkan percepatan roll dengan menggunakan fungsi `wb_motor_set_position()`. Posisi motor diubah sesuai dengan nilai -0.115 dikali dengan percepatan roll.

```
wb_motor_set_position(camera_pitch_motor, -0.1 * pitch_acceleration);
```

Baris ini mengatur posisi motor kamera pitch berdasarkan percepatan pitch dengan menggunakan fungsi `wb_motor_set_position()`. Posisi motor diubah sesuai dengan nilai -0.1 dikali dengan percepatan pitch.

Bagian selanjutnya menangani input dari keyboard dan mengubahnya menjadi gangguan dalam algoritma stabilisasi, menghitung input roll, pitch, yaw, dan vertical.

Dan pada bagian terakhir berfungsi sebagai mengaktifkan motor dengan mempertimbangkan semua input yang telah dihitung sebelumnya.



Kode di atas terdiri dari dua pernyataan:

`wb_robot_cleanup();`

Pernyataan ini digunakan untuk membersihkan dan menutup semua sumber daya yang telah digunakan oleh Webots untuk robot. Fungsi ini harus dipanggil sebelum program berakhir untuk memastikan semua sumber daya dibebaskan dengan benar.

`return EXIT_SUCCESS;`

Pernyataan ini mengembalikan nilai keluaran program yang menandakan bahwa program telah berhasil dieksekusi dan selesai berjalan. Nilai `EXIT_SUCCESS` adalah konstanta yang didefinisikan dalam library `stdlib.h` dan menunjukkan bahwa program selesai tanpa adanya kesalahan.

Dengan memanggil `wb_robot_cleanup()` dan mengembalikan `EXIT_SUCCESS`, program dapat diakhiri dengan benar dan mengindikasikan bahwa program berjalan dengan sukses tanpa ada kesalahan.