Robotics Techical Report

“Hacking Webot using E-puck line demo robot ”

Universitas Telkom

Oleh:

Muhammad Haidar Abdul Jabbar

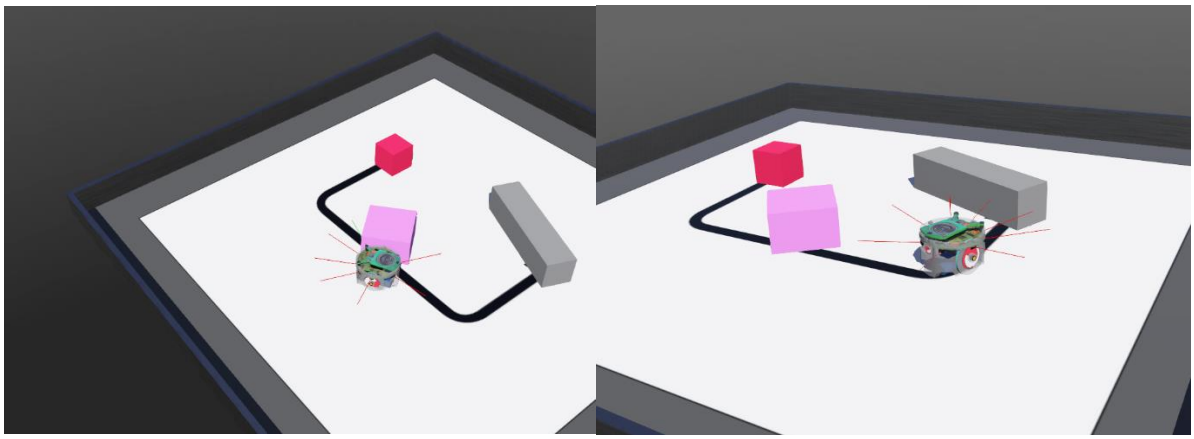
1103202071

Program Studi Teknik Komputer

Fakultas Teknik Elektro

Universitas Telkom

2022/2023



Puck line adalah variasi dari robot E-Puck yang dirancang khusus untuk mengikuti garis atau jalur yang telah ditentukan. Puck line memungkinkannya untuk melacak dan mengikuti garis. Dalam mengikuti garis atau jalur yang telah ditentukan, robot Puck Line menggunakan sensor infra merah untuk mendeteksi perbedaan warna antara garis dan area di sekitarnya. Sensor infra merah ini berfungsi untuk mengukur reflektivitas permukaan dan memungkinkan robot untuk mengenali garis dengan akurasi tinggi.

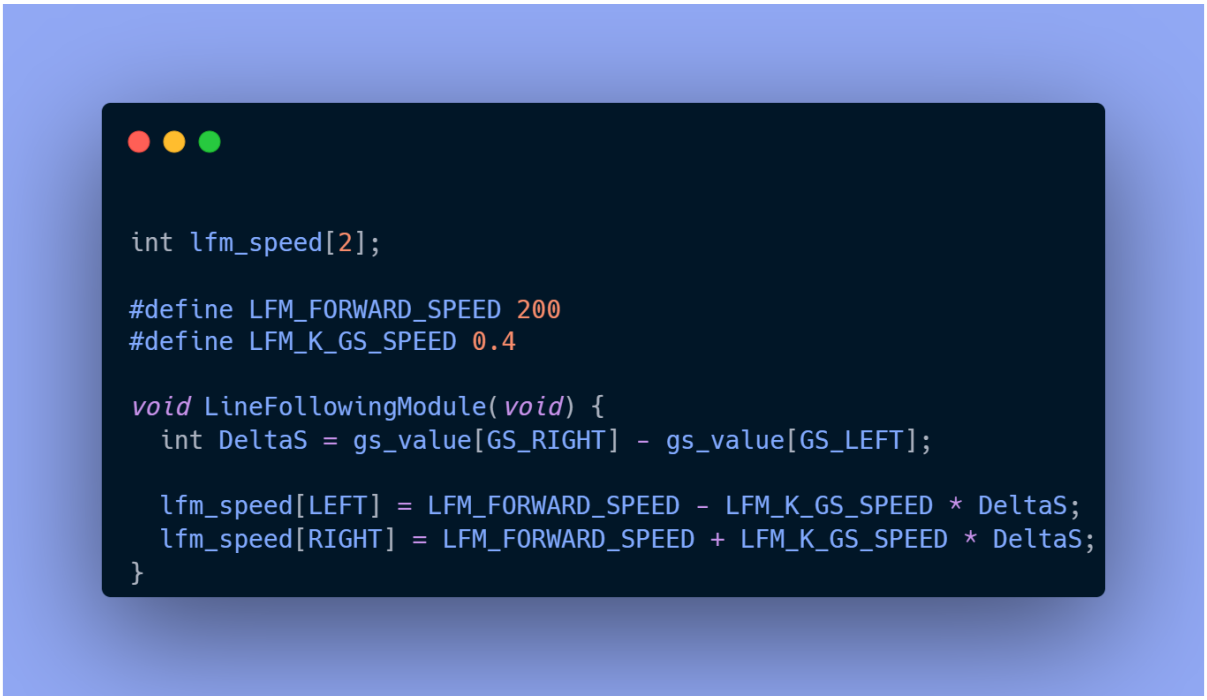
Ketika robot Puck Line mengenali garis, maka motor yang terpasang pada robot akan diaktifkan dan digerakkan secara otomatis oleh program yang telah diprogramkan sebelumnya. Gerakan motor inilah yang akan memungkinkan robot untuk mengikuti garis atau jalur yang telah ditentukan dengan akurasi tinggi.

Untuk memperbaiki akurasi dan stabilitas robot dalam mengikuti garis atau jalur, beberapa algoritma kontrol digunakan dalam program yang diprogramkan pada robot. Dalam pengaturan kontrol yang lebih canggih, robot Puck Line dapat menyesuaikan kecepatannya dengan perubahan jalur atau garis yang terdeteksi oleh sensor infra merah, sehingga robot dapat mengikuti garis atau jalur dengan lebih presisi dan kecepatan yang lebih konsisten.

Penjelasan Souce Code:

```c
#include <stdio.h>
#include <webots/distance_sensor.h>
#include <webots/led.h>
#include <webots/light_sensor.h>
#include <webots/motor.h>
#include <webots/robot.h>

// Global defines
#define TRUE 1
#define FALSE 0
#define NO_SIDE -1
#define LEFT 0
#define RIGHT 1
#define WHITE 0
#define BLACK 1
#define SIMULATION 0  // for wb_robot_get_mode() function
#define REALITY 2     // for wb_robot_get_mode() function
#define TIME_STEP 32  // [ms]

// 8 IR proximity sensors
#define NB_DIST_SENS 8
#define PS_RIGHT_00 0
#define PS_RIGHT_45 1
#define PS_RIGHT_90 2
#define PS_RIGHT_REAR 3
#define PS_LEFT_REAR 4
#define PS_LEFT_90 5
#define PS_LEFT_45 6
#define PS_LEFT_00 7
const int PS_OFFSET_SIMULATION[NB_DIST_SENS] = {300, 300, 300, 300, 300, 300, 300,
300};
const int PS_OFFSET_REALITY[NB_DIST_SENS] = {480, 170, 320, 500, 600, 680, 210, 640};
WbDeviceTag ps[NB_DIST_SENS]; /* proximity sensors */
int ps_value[NB_DIST_SENS] = {0, 0, 0, 0, 0, 0, 0, 0};

// 3 IR ground color sensors
#define NB_GROUND_SENS 3
#define GS_WHITE 900
#define GS_LEFT 0
#define GS_CENTER 1
#define GS_RIGHT 2
WbDeviceTag gs[NB_GROUND_SENS]; /* ground sensors */
unsigned short gs_value[NB_GROUND_SENS] = {0, 0, 0};

// Motors
WbDeviceTag left_motor, right_motor;

// LEDs
#define NB_LEDS 8
WbDeviceTag led[NB_LEDS];
```

Pada bagian ini diperlukan dalam melakukan import library yang diperlukan dan digunakan dalam menginisialisasi komponen yang diperlukan,seperti led,sensor,dsb

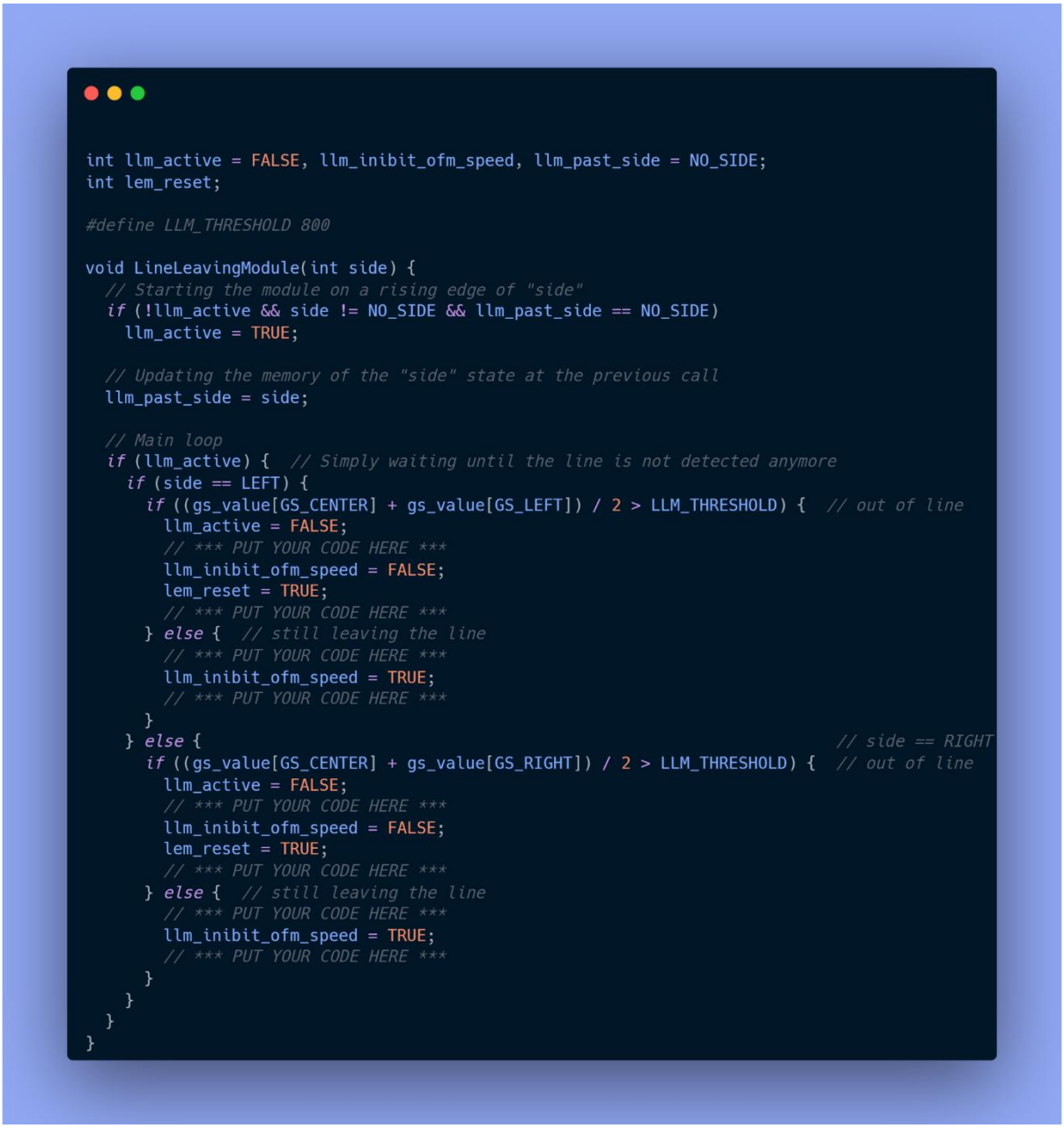


```
int lfm_speed[2];

#define LFM_FORWARD_SPEED 200
#define LFM_K_GS_SPEED 0.4

void LineFollowingModule(void) {
  int DeltaS = gs_value[GS_RIGHT] - gs_value[GS_LEFT];

  lfm_speed[LEFT] = LFM_FORWARD_SPEED - LFM_K_GS_SPEED * DeltaS;
  lfm_speed[RIGHT] = LFM_FORWARD_SPEED + LFM_K_GS_SPEED * DeltaS;
}
```

Kode ini implementasi yang digunakan dalam mengikuti garis hitam di tanah. Kecepatan output disimpan lfm_speed [LEFT] lfm_speed [RIGHT].

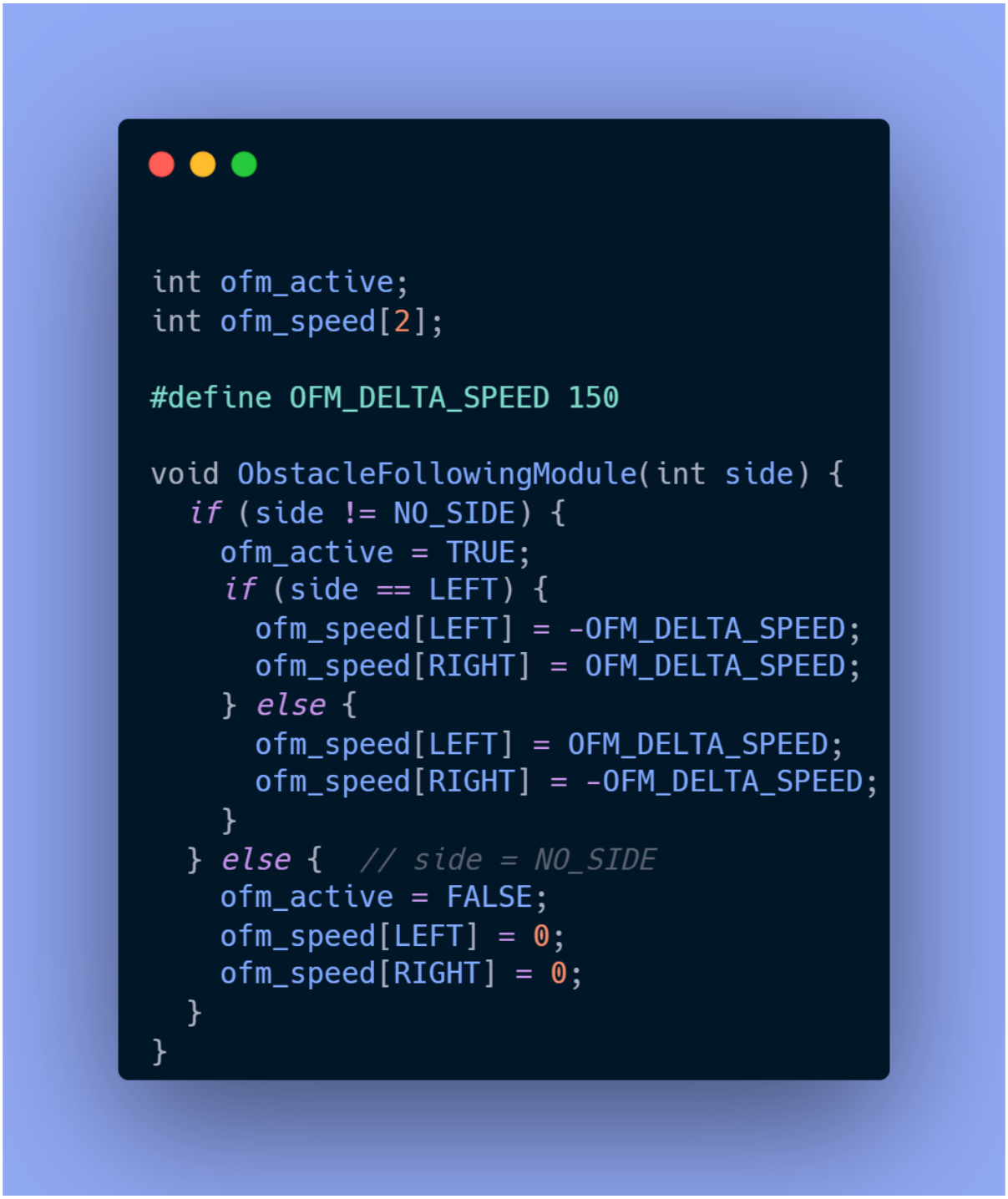
```c
int oam_active, oam_reset;
int oam_speed[2];
int oam_side = NO_SIDE;

#define OAM_OBST_THRESHOLD 100
#define OAM_FORWARD_SPEED 150
#define OAM_K_PS_90 0.2
#define OAM_K_PS_45 0.9
#define OAM_K_PS_00 1.2
#define OAM_K_MAX_DELTAS 600

void ObstacleAvoidanceModule(void) {
  int max_ds_value, i;
  int Activation[] = {0, 0};

  // Module RESET
  if (oam_reset) {
    oam_active = FALSE;
    oam_side = NO_SIDE;
  }
  oam_reset = 0;

  // Determine the presence and the side of an obstacle
  max_ds_value = 0;
  for (i = PS_RIGHT_00; i <= PS_RIGHT_45; i++) {
    if (max_ds_value < ps_value[i])
      max_ds_value = ps_value[i];
    Activation[RIGHT] += ps_value[i];
  }
  for (i = PS_LEFT_45; i <= PS_LEFT_00; i++) {
    if (max_ds_value < ps_value[i])
      max_ds_value = ps_value[i];
    Activation[LEFT] += ps_value[i];
  }
  if (max_ds_value > OAM_OBST_THRESHOLD)
    oam_active = TRUE;

  if (oam_active && oam_side == NO_SIDE)  // check for side of obstacle only when not already detected
  {
    if (Activation[RIGHT] > Activation[LEFT])
      oam_side = RIGHT;
    else
      oam_side = LEFT;
  }

  // Forward speed
  oam_speed[LEFT] = OAM_FORWARD_SPEED;
  oam_speed[RIGHT] = OAM_FORWARD_SPEED;

  // Go away from obstacle
  if (oam_active) {
    int DeltaS = 0;
    // The rotation of the robot is determined by the location and the side of the obstacle
    if (oam_side == LEFT) {
      //(((ps_value[PS_LEFT_90]-PS_OFFSET)<0)?0:(ps_value[PS_LEFT_90]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_90 * ps_value[PS_LEFT_90]);
      //(((ps_value[PS_LEFT_45]-PS_OFFSET)<0)?0:(ps_value[PS_LEFT_45]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_45 * ps_value[PS_LEFT_45]);
      //(((ps_value[PS_LEFT_00]-PS_OFFSET)<0)?0:(ps_value[PS_LEFT_00]-PS_OFFSET)));
      DeltaS -= (int)(OAM_K_PS_00 * ps_value[PS_LEFT_00]);
    } else {  // oam_side == RIGHT
      //(((ps_value[PS_RIGHT_90]-PS_OFFSET)<0)?0:(ps_value[PS_RIGHT_90]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_90 * ps_value[PS_RIGHT_90]);
      //(((ps_value[PS_RIGHT_45]-PS_OFFSET)<0)?0:(ps_value[PS_RIGHT_45]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_45 * ps_value[PS_RIGHT_45]);
      //(((ps_value[PS_RIGHT_00]-PS_OFFSET)<0)?0:(ps_value[PS_RIGHT_00]-PS_OFFSET)));
      DeltaS += (int)(OAM_K_PS_00 * ps_value[PS_RIGHT_00]);
    }
    if (DeltaS > OAM_K_MAX_DELTAS)
      DeltaS = OAM_K_MAX_DELTAS;
    if (DeltaS < -OAM_K_MAX_DELTAS)
      DeltaS = -OAM_K_MAX_DELTAS;

    // Set speeds
    oam_speed[LEFT] -= DeltaS;
    oam_speed[RIGHT] += DeltaS;
  }
}
```

Routine OAM pertama-tama mendeteksi rintangan di depan robot, lalu merekam sisi di"oam_side" dan menghindari penghalang/obstacle yang terdeteksi sensor jarak yang terdapat pada robot.



```
int llm_active = FALSE, llm_inibit_ofm_speed, llm_past_side = NO_SIDE;
int lem_reset;

#define LLM_THRESHOLD 800

void LineLeavingModule(int side) {
  // Starting the module on a rising edge of "side"
  if (!llm_active && side != NO_SIDE && llm_past_side == NO_SIDE)
    llm_active = TRUE;

  // Updating the memory of the "side" state at the previous call
  llm_past_side = side;

  // Main loop
  if (llm_active) {  // Simply waiting until the line is not detected anymore
    if (side == LEFT) {
      if ((gs_value[GS_CENTER] + gs_value[GS_LEFT]) / 2 > LLM_THRESHOLD) {  // out of line
        llm_active = FALSE;
        // *** PUT YOUR CODE HERE ***
        llm_inibit_ofm_speed = FALSE;
        lem_reset = TRUE;
        // *** PUT YOUR CODE HERE ***
      } else {  // still leaving the line
        // *** PUT YOUR CODE HERE ***
        llm_inibit_ofm_speed = TRUE;
        // *** PUT YOUR CODE HERE ***
      }
    } else {                                                              // side == RIGHT
      if ((gs_value[GS_CENTER] + gs_value[GS_RIGHT]) / 2 > LLM_THRESHOLD) {  // out of line
        llm_active = FALSE;
        // *** PUT YOUR CODE HERE ***
        llm_inibit_ofm_speed = FALSE;
        lem_reset = TRUE;
        // *** PUT YOUR CODE HERE ***
      } else {  // still leaving the line
        // *** PUT YOUR CODE HERE ***
        llm_inibit_ofm_speed = TRUE;
        // *** PUT YOUR CODE HERE ***
      }
    }
  }
}
```

Karena tidak memiliki output, routine ini belum sepenuhnya selesai. Dirancang untuk memantau kondisi ketika robot meninggalkan posisi.Yang kemudian akan melacak dan memberi sinyal ke modul lain.



```
int ofm_active;
int ofm_speed[2];

#define OFM_DELTA_SPEED 150

void ObstacleFollowingModule(int side) {
  if (side != NO_SIDE) {
    ofm_active = TRUE;
    if (side == LEFT) {
      ofm_speed[LEFT] = -OFM_DELTA_SPEED;
      ofm_speed[RIGHT] = OFM_DELTA_SPEED;
    } else {
      ofm_speed[LEFT] = OFM_DELTA_SPEED;
      ofm_speed[RIGHT] = -OFM_DELTA_SPEED;
    }
  } else {  // side = NO_SIDE
    ofm_active = FALSE;
    ofm_speed[LEFT] = 0;
    ofm_speed[RIGHT] = 0;
  }
}
```

Kode ini memberikan kecenderungan robot untuk mengarahkan ke sampingditunjukkan oleh argumennya "sisi". Saat digunakan dalam persaingan dengan OAM hal ini akan memunculkan objek yang mengikuti perilaku. Kecepatan keluaran adalahdisimpan di ofm_speed[LEFT] dan ofm_speed[RIGHT].

```
int lem_active;
int lem_speed[2];
int lem_state, lem_black_counter;
int cur_op_gs_value, prev_op_gs_value;

#define LEM_FORWARD_SPEED 100
#define LEM_K_GS_SPEED 0.5
#define LEM_THRESHOLD 500

#define LEM_STATE_STANDBY 0
#define LEM_STATE_LOOKING_FOR_LINE 1
#define LEM_STATE_LINE_DETECTED 2
#define LEM_STATE_ON_LINE 3

void LineEnteringModule(int side) {
  int Side, OpSide, GS_Side, GS_OpSide;

  // Module reset
  if (lem_reset)
    lem_state = LEM_STATE_LOOKING_FOR_LINE;
  lem_reset = FALSE;

  // Initialization
  lem_speed[LEFT] = LEM_FORWARD_SPEED;
  lem_speed[RIGHT] = LEM_FORWARD_SPEED;
  if (side == LEFT) {  // if obstacle on left side -> enter line rightward
    Side = RIGHT;      // line entering direction
    OpSide = LEFT;
    GS_Side = GS_RIGHT;
    GS_OpSide = GS_LEFT;
  } else {        // if obstacle on left side -> enter line leftward
    Side = LEFT;  // line entering direction
    OpSide = RIGHT;
    GS_Side = GS_LEFT;
    GS_OpSide = GS_RIGHT;
  }

  // Main loop (state machine)
  switch (lem_state) {
    case LEM_STATE_STANDBY:
      lem_active = FALSE;
      break;
    case LEM_STATE_LOOKING_FOR_LINE:
      if (gs_value[GS_Side] < LEM_THRESHOLD) {
        lem_active = TRUE;
        // set speeds for entering line
        lem_speed[OpSide] = LEM_FORWARD_SPEED;
        lem_speed[Side] = LEM_FORWARD_SPEED;  // - LEM_K_GS_SPEED * gs_value[GS_Side];
        lem_state = LEM_STATE_LINE_DETECTED;
        // save ground sensor value
        if (gs_value[GS_OpSide] < LEM_THRESHOLD) {
          cur_op_gs_value = BLACK;
          lem_black_counter = 1;
        } else {
          cur_op_gs_value = WHITE;
          lem_black_counter = 0;
        }
        prev_op_gs_value = cur_op_gs_value;
      }
      break;
    case LEM_STATE_LINE_DETECTED:
      // save the oposite ground sensor value
      if (gs_value[GS_OpSide] < LEM_THRESHOLD) {
        cur_op_gs_value = BLACK;
        lem_black_counter++;
      } else
        cur_op_gs_value = WHITE;
      // detect the falling edge BLACK->WHITE
      if (prev_op_gs_value == BLACK && cur_op_gs_value == WHITE) {
        lem_state = LEM_STATE_ON_LINE;
        lem_speed[OpSide] = 0;
        lem_speed[Side] = 0;
      } else {
        prev_op_gs_value = cur_op_gs_value;
        // set speeds for entering line
        lem_speed[OpSide] = LEM_FORWARD_SPEED + LEM_K_GS_SPEED * (GS_WHITE - gs_value[GS_Side]);
        lem_speed[Side] = LEM_FORWARD_SPEED - LEM_K_GS_SPEED * (GS_WHITE - gs_value[GS_Side]);
      }
      break;
    case LEM_STATE_ON_LINE:
      oam_reset = TRUE;
      lem_active = FALSE;
      lem_state = LEM_STATE_STANDBY;
      break;
  }
}
```

Tujuan Kode ini adalah untuk menangani ketika kondisi robot harus masuk kembali ke lintasan (setelah melewati hambatan). Inputnya adalah sensor arde lateral,argumen "sisi" yang menentukan arah yang harus diikuti robot saat mendeteksi garis hitam, dan Kecepatan keluaran disimpan dalam lem_speed[LEFT] dan lem_speed[RIGHT].

```c
int main() {
  int ps_offset[NB_DIST_SENS] = {0, 0, 0, 0, 0, 0, 0, 0}, i, speed[2], Mode = 1;
  int oam_ofm_speed[2];

  /* intialize Webots */
  wb_robot_init();

  /* initialization */
  char name[20];
  for (i = 0; i < NB_LEDS; i++) {
    sprintf(name, "led%d", i);
    led[i] = wb_robot_get_device(name); /* get a handler to the sensor */
  }
  for (i = 0; i < NB_DIST_SENS; i++) {
    sprintf(name, "ps%d", i);
    ps[i] = wb_robot_get_device(name); /* proximity sensors */
    wb_distance_sensor_enable(ps[i], TIME_STEP);
  }
  for (i = 0; i < NB_GROUND_SENS; i++) {
    sprintf(name, "gs%d", i);
    gs[i] = wb_robot_get_device(name); /* ground sensors */
    wb_distance_sensor_enable(gs[i], TIME_STEP);
  }
  // motors
  left_motor = wb_robot_get_device("left wheel motor");
  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);
  wb_motor_set_velocity(right_motor, 0.0);

  for (;;) {  // Main loop
    // Run one simulation step
    wb_robot_step(TIME_STEP);

    // Reset all BB variables when switching from simulation to real robot and back
    if (Mode != wb_robot_get_mode()) {
      oam_reset = TRUE;
      llm_active = FALSE;
      llm_past_side = NO_SIDE;
      ofm_active = FALSE;
      lem_active = FALSE;
      lem_state = LEM_STATE_STANDBY;
      Mode = wb_robot_get_mode();
      if (Mode == SIMULATION) {
        for (i = 0; i < NB_DIST_SENS; i++)
          ps_offset[i] = PS_OFFSET_SIMULATION[i];
        wb_motor_set_velocity(left_motor, 0);
        wb_motor_set_velocity(right_motor, 0);
        wb_robot_step(TIME_STEP);  // Just run one step to make sure we get correct sensor values
        printf("\n\n\nSwitching to SIMULATION and reseting all BB variables.\n\n");
      } else if (Mode == REALITY) {
        for (i = 0; i < NB_DIST_SENS; i++)
          ps_offset[i] = PS_OFFSET_REALITY[i];
        wb_motor_set_velocity(left_motor, 0);
        wb_motor_set_velocity(right_motor, 0);
        wb_robot_step(TIME_STEP);  // Just run one step to make sure we get correct sensor values
        printf("\n\n\nSwitching to REALITY and reseting all BB variables.\n\n");
      }
    }

    // read sensors value
    for (i = 0; i < NB_DIST_SENS; i++)
      ps_value[i] = (((int)wb_distance_sensor_get_value(ps[i]) - ps_offset[i]) < 0) ?
                    0 :
                    ((int)wb_distance_sensor_get_value(ps[i]) - ps_offset[i]);
    for (i = 0; i < NB_GROUND_SENS; i++)
      gs_value[i] = wb_distance_sensor_get_value(gs[i]);

    // Speed initialization
    speed[LEFT] = 0;
    speed[RIGHT] = 0;

    // *** START OF SUBSUMPTION ARCHITECTURE ***

    // LFM - Line Following Module
    LineFollowingModule();

    speed[LEFT] = lfm_speed[LEFT];
    speed[RIGHT] = lfm_speed[RIGHT];

    // OAM - Obstacle Avoidance Module
    ObstacleAvoidanceModule();

    // LLM - Line Leaving Module
    LineLeavingModule(oam_side);

    // OFM - Obstacle Following Module
    ObstacleFollowingModule(oam_side);

    // Inibit A
    if (llm_inibit_ofm_speed) {
      ofm_speed[LEFT] = 0;
      ofm_speed[RIGHT] = 0;
    }

    // Sum A
    oam_ofm_speed[LEFT] = oam_speed[LEFT] + ofm_speed[LEFT];
    oam_ofm_speed[RIGHT] = oam_speed[RIGHT] + ofm_speed[RIGHT];

    // Suppression A
    if (oam_active || ofm_active) {
      speed[LEFT] = oam_ofm_speed[LEFT];
      speed[RIGHT] = oam_ofm_speed[RIGHT];
    }

    // LEM - Line Entering Module
    LineEnteringModule(oam_side);

    // Suppression B
    if (lem_active) {
      speed[LEFT] = lem_speed[LEFT];
      speed[RIGHT] = lem_speed[RIGHT];
    }

    // *** END OF SUBSUMPTION ARCHITECTURE ***

    // Debug display
    printf("OAM %d side %d   LLM %d inibitA %d   OFM %d   LEM %d state %d oam_reset %d\n", oam_active,
oam_side, llm_active,
           llm_inibit_ofm_speed, ofm_active, lem_active, lem_state, oam_reset);

    // Set wheel speeds
    wb_motor_set_velocity(left_motor, 0.00628 * speed[LEFT]);
    wb_motor_set_velocity(right_motor, 0.00628 * speed[RIGHT]);
  }
  return 0;
}
```

Merupakan Main function pada program ,yang berfungsi dalam menginisialisasi robot, membaca nilai sensor, dan menjalankan loop arsitektur subsumption, yang terdiri dari modul berbeda yang memproses informasi sensor dan mengontrol gerakan robot berdasarkan perilaku yang berbeda, seperti mengikuti garis, menghindari rintangan, masuk dan keluar garis, dan kendala berikut. Program ini juga menyertakan kode debug untuk menampilkan informasi tentang status berbagai modul selama looping.