МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

Отчет

по лабораторной работе №3

по дисциплине «Аппаратное и программное обеспечение роботизированных систем»

«Программирование алгоритмов управления роботов в Webots»

РУКОВОДИТЕЛЬ:

_____ Гай В.Е.

СТУДЕНТ:

_____ Ширшов А.А.

_____ Юрчук М.С.

19-В-1

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2022

Цель: получение навыков работы с алгоримами управления роботами.

Задание: выполнить вариант, записать видео работы.

Вариант 2. Движение по линии

Робот должен выполнить движение по линии, от начала до конца;

Робот: e-puck (https://cyberbotics.com/doc/guide/epuck),

Структура сцены: размер 5*5 метров (ограничена стенами), линия, вдоль которой движется должна быть сплошной с прерывистыми участками, размер разрывов должен быть не больше половины размера робота, на линии также должны находиться препятствия, которые робот должен обходить.

Пояснение к коду:

После инициализации датчиков программа переходит в бесконечный цикл. В этом цикле происходит последовательный вызов функций, которые из себя представляют так называемую машину состояний:

• модуль следования по линии — в зависимости от состояний датчиков света робот будет поворачивать в противоположную сторону (например, белый на правом — поворот на лево)

• модуль обхода препятстви — если впереди робота находится препятствие, то происходит определение стороны, в которой находится препятствие, высчитывание разницы и ее сложение со скоростями движения колес.

• модуль потери линии — робот в случае потери линии модуль уведомит с помощью флагов остальные модули о потери линии

• модуль следования по препятствию — в случае обнаружения припятствия не позволит роботу отдалиться от препятствия (из-за модуля обхода препятствия), таким образом робот будет двигаться по краю препятствия

• модуль входа в линию — цель модуля — перехватить момент обнаружения линии и начать двигаться по ней

Исходный код:

```
#include <algorithm>
#include <webots/Robot.hpp>
#include <webots/Motor.hpp>
#include <webots/DistanceSensor.hpp>

// All the webots classes are defined in the "webots" namespace
using namespace webots;

// Global defines
constexpr auto NO_SIDE = -1;
constexpr auto LEFT = 0;
constexpr auto RIGHT = 1;
constexpr auto WHITE = 0;
constexpr auto BLACK = 1;
constexpr auto TIME_STEP = 32;

// 8 IR proximity sensors
constexpr auto NB_DIST_SENS = 8;
int ps_value[ NB_DIST_SENS ] = { 0, 0, 0, 0, 0, 0, 0, 0 };

// 3 IR ground color sensors
constexpr auto NB_GROUND_SENS = 3;
constexpr auto GS_WHITE = 900;
constexpr auto GS_LEFT = 0;
constexpr auto GS_CENTER = 1;
constexpr auto GS_RIGHT = 2;
int gs_value[ NB_GROUND_SENS ] = { 0, 0, 0 };

// Line Following Module
constexpr auto LFM_FORWARD_SPEED = 200;
constexpr auto LFM_K_GS_SPEED = 0.4;
int lfm_speed[ 2 ]{};

// Obstacle Avoidance Module
constexpr auto OAM_OBST_THRESHOLD = 100;
constexpr auto OAM_FORWARD_SPEED = 150;
constexpr auto OAM_K_PS_90 = 0.2;
```

```
constexpr auto OAM_K_PS_45 = 0.9;
constexpr auto OAM_K_PS_00 = 1.2;
constexpr auto OAM_K_MAX_DELTAS = 600;
bool oam_active, oam_reset;
int oam_speed[ 2 ]{};
int oam_side = NO_SIDE;

// Line Leaving Module
constexpr auto LLM_THRESHOLD = 800;
bool llm_active = false, llm_inibit_ofm_speed, lem_reset;
int llm_past_side = NO_SIDE;

// Obstacle Following Module
constexpr auto OFM_DELTA_SPEED = 150;
bool ofm_active;
int ofm_speed[ 2 ]{};

// Line Entering Module
constexpr auto LEM_FORWARD_SPEED = 100;
constexpr auto LEM_THRESHOLD = 500;
constexpr auto LEM_STATE_STANDBY = 0;
constexpr auto LEM_STATE_LOOKING_FOR_LINE = 1;
constexpr auto LEM_STATE_LINE_DETECTED = 2;
constexpr auto LEM_STATE_ON_LINE = 3;
bool lem_active;
int lem_speed[ 2 ]{};
int lem_state;
int cur_op_gs_value, prev_op_gs_value;

enum PS_SIDE {
    RIGHT_00 = 0,
    RIGHT_45,
    RIGHT_90,
    RIGHT_REAR,
    LEFT_REAR,
    LEFT_90,
    LEFT_45,
    LEFT_00
};

////////////////////////////////////////////
// LFM - Line Following Module
//
// This module implements a very simple, Braitenberg-like behavior in order
// to follow a black line on the ground. Output speeds are stored in
// lfm_speed[LEFT] and lfm_speed[RIGHT].
void LineFollowingModule( void ) {
    int DeltaS = gs_value[ GS_RIGHT ] - gs_value[ GS_LEFT ];

    lfm_speed[ LEFT ] = static_cast< int >( LFM_FORWARD_SPEED - LFM_K_GS_SPEED * DeltaS );
    lfm_speed[ RIGHT ] = static_cast< int >( LFM_FORWARD_SPEED + LFM_K_GS_SPEED * DeltaS );
}

////////////////////////////////////////////
// OAM - Obstacle Avoidance Module
//
// The OAM routine first detects obstacles in front of the robot, then records
// their side in "oam_side" and avoid the detected obstacle by
// turning away according to very simple weighted connections between
// proximity sensors and motors. "oam_active" becomes active when as soon as
// an object is detected and "oam_reset" inactivates the module and set
// "oam_side" to NO_SIDE. Output speeds are in oam_speed[LEFT] and oam_speed[RIGHT].
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 = false;

    // Determine the presence and the side of an obstacle
    max_ds_value = 0;
    for ( i = PS_SIDE::RIGHT_00; i <= PS_SIDE::RIGHT_45; i++ ) {
        if ( max_ds_value < ps_value[ i ] )
            max_ds_value = ps_value[ i ];

        Activation[ RIGHT ] += ps_value[ i ];
    }
    for ( i = PS_SIDE::LEFT_45; i <= PS_SIDE::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 ) {
            DeltaS -= static_cast< int >( OAM_K_PS_90 * ps_value[ PS_SIDE::LEFT_90 ] );
            DeltaS -= static_cast< int >( OAM_K_PS_45 * ps_value[ PS_SIDE::LEFT_45 ] );
            DeltaS -= static_cast< int >( OAM_K_PS_00 * ps_value[ PS_SIDE::LEFT_00 ] );

            std::cout << "= OAM | ps[5]: " << ps_value[ PS_SIDE::LEFT_90 ] << " | "
                << " ps[6]: " << ps_value[ PS_SIDE::LEFT_45 ] << " ps[7]: " << ps_value[ PS_SIDE::LEFT_00 ] << std::endl;
        }
        else {  // oam_side == RIGHT
            DeltaS += static_cast< int >( OAM_K_PS_90 * ps_value[ PS_SIDE::RIGHT_90 ] );
            DeltaS += static_cast< int >( OAM_K_PS_45 * ps_value[ PS_SIDE::RIGHT_45 ] );
            DeltaS += static_cast< int >( OAM_K_PS_00 * ps_value[ PS_SIDE::RIGHT_00 ] );

            std::cout << "= OAM | ps[2]: " << ps_value[ PS_SIDE::RIGHT_90 ] << " | "
                << " ps[1]: " << ps_value[ PS_SIDE::RIGHT_45 ] << " ps[0]: " << ps_value[ PS_SIDE::RIGHT_00 ] <<
std::endl;
        }

        DeltaS = std::clamp( DeltaS, -OAM_K_MAX_DELTAS, OAM_K_MAX_DELTAS );

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

//////////////////////////////////////////
// LLM - Line Leaving Module
//
// It has been designed to monitor the moment while the robot is leaving the
// track and signal to other modules some related events. It becomes active
// whenever the "side" variable displays a rising edge (changing from -1 to 0 or 1).
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_inibit_ofm_speed = false;
                llm_active = false;
                lem_reset = true;
            }
            else {  // still leaving the line
                llm_inibit_ofm_speed = true;
            }
        }
        else {                                                  // side == RIGHT
            if ( ( gs_value[ GS_CENTER ] + gs_value[ GS_RIGHT ] ) / 2 > LLM_THRESHOLD ) {  // out of line
                llm_inibit_ofm_speed = false;
                llm_active = false;
                lem_reset = true;
            }
            else {  // still leaving the line
                llm_inibit_ofm_speed = true;
            }
        }
    }
```

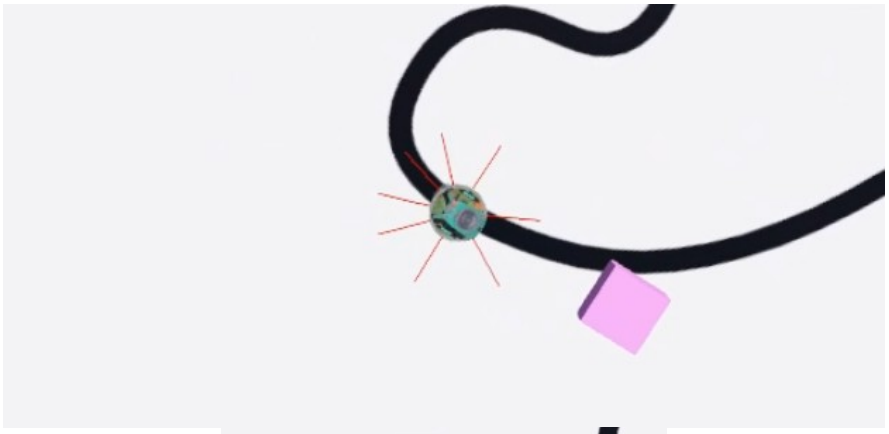
```
}

//////////////////////////////////////////
// OFM - Obstacle Following Module
//
// This function just gives the robot a tendency to steer toward the side
// indicated by its argument "side". When used in competition with OAM it
// gives rise to an object following behavior. The output speeds are
// stored in ofm_speed[LEFT] and ofm_speed[RIGHT].
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;
    }
}

//////////////////////////////////////////
// LEM - Line Entering Module
//
// Its purpose is to handle the moment when
// the robot must re-enter the track (after having by-passed
// an obstacle, e.g.). It is organized like a state machine, which state is
// stored in "lem_state" (see LEM_STATE_STANDBY and following #defines).
// The inputs are the two lateral ground sensors, the argument "side"
// which determines the direction that the robot has to follow when detecting
// a black line, and the variable "lem_reset" that resets the state to
// standby. The output speeds are stored in lem_speed[LEFT] and
// lem_speed[RIGHT].
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_state = LEM_STATE_LINE_DETECTED;

            // save ground sensor value
            if ( gs_value[ GS_OpSide ] < LEM_THRESHOLD ) {
                cur_op_gs_value = BLACK;
            }
            else {
```

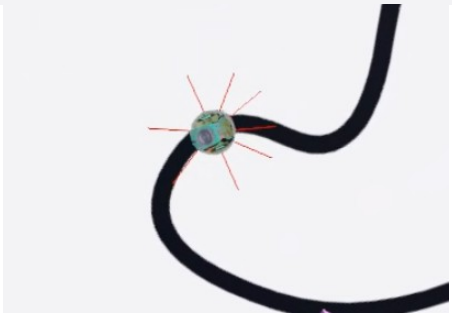
```
            cur_op_gs_value = WHITE;
        }

        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;
    }
    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_STATE_LOOKING_FOR_LINE * ( GS_WHITE -
gs_value[ GS_Side ] );
        lem_speed[ Side ] = LEM_FORWARD_SPEED - LEM_STATE_LOOKING_FOR_LINE * ( GS_WHITE - gs_value[
GS_Side ] );
    }
    break;
  case LEM_STATE_ON_LINE:
    oam_reset = true;
    lem_active = false;
    lem_state = LEM_STATE_STANDBY;
    break;
  }
}

int main( int argc, char** argv ) {
  auto side_to_str = [ ]( unsigned side) {
    switch ( side )         {
    case LEFT:
        return "left";
        break;
    case RIGHT:
        return "right";
        break;
    case NO_SIDE:
        return "no side";
        break;
    default:
        return "none";
        break;
    }
  };

  // Create the Robot instance.
  Robot* robot = new Robot( );

  DistanceSensor* ps[ NB_DIST_SENS ]; // proximity sensors
  DistanceSensor* gs[ NB_GROUND_SENS ]; // ground sensors

  // Initialization
  for ( int i = 0; i < NB_DIST_SENS; i++ ) {
    ps[ i ] = robot->getDistanceSensor( "ps" + std::to_string( i ) ); // proximity sensors
    ps[ i ]->enable( TIME_STEP );
  }
  for ( int i = 0; i < NB_GROUND_SENS; i++ ) {
    gs[ i ] = robot->getDistanceSensor( "gs" + std::to_string( i ) ); // ground sensors
    gs[ i ]->enable( TIME_STEP );
  }

  // Motors
  Motor *left_motor = robot->getMotor( "left wheel motor" );
  Motor *right_motor = robot->getMotor( "right wheel motor" );

  left_motor->setPosition( INFINITY );
  right_motor->setPosition( INFINITY );

  // Reset all BB variables
  oam_reset = true;
  llm_active = false;
  llm_past_side = NO_SIDE;
  ofm_active = false;
  lem_active = false;
  lem_state = LEM_STATE_STANDBY;
```

```
    left_motor->setVelocity( 0. );
    right_motor->setVelocity( 0. );

    int speed[ 2 ]{ 0, 0 };
    int oam_ofm_speed[ 2 ]{};

    for (;;) {  // Main loop
        // Run one simulation step
        robot->step( TIME_STEP );

        // read sensors value
        for ( int i = 0; i < NB_DIST_SENS; i++ )
            ps_value[ i ] = std::clamp( static_cast< int >( ps[ i ]->getValue( ) - 300 ), 0, 9999 );

        for ( int i = 0; i < NB_GROUND_SENS; i++ )
            gs_value[ i ] = static_cast< int >( gs[ i ]->getValue( ) );

        // 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 ];

            std::cout << "Left speed: " << speed[ LEFT ] << " | right speed: " << speed[ RIGHT ] << std::endl;
            std::cout << "Left OFM speed: " << ofm_speed[ LEFT ] << " | right OFM speed: " << ofm_speed[ RIGHT ] <<
std::endl;
            std::cout << "Left OAM speed: " << oam_speed[ LEFT ] << " | right OAM speed: " << oam_speed[ RIGHT ] <<
std::endl;
        }

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

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

        // Debug display
        std::cout << "OAM: " << oam_active << ", side: " << side_to_str( oam_side )
            << " | LLM: " << llm_active << ", inibitA: " << llm_inibit_ofm_speed
            << " | OFM: " << ofm_active << " | LEM: " << lem_active << ", state: " << lem_state
            << " | OAM reset: " << oam_reset << std::endl;

        // Set wheel speeds
        left_motor->setVelocity( std::max( 0., std::min( ( 0.00628 * speed[ LEFT ] ), 6.28 ) ) );
        right_motor->setVelocity( std::max( 0., std::min( ( 0.00628 * speed[ RIGHT ] ), 6.28 ) ) );
    }

    // Enter here exit cleanup code.

    delete robot;
    return 0;
}
```

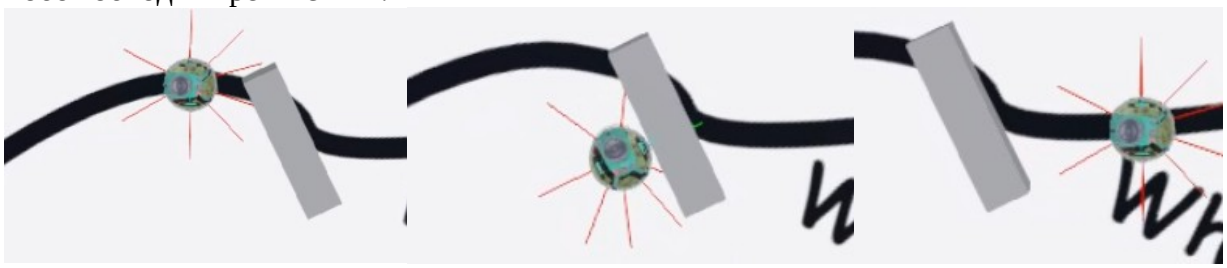
Результат:

Робот следует по линии:





Робот обходит препятствия:



Робот едет по прерывающейся линии:

Вывод: в результате выполнения данной лабораторной работы были получены алгоритмы для управления роботов с помощью обратной связи (датчиков). Был разработан алгоритм, по которому робот может ехать по линии и объезжать препрятствия.