ROBÓTICA INDUSTRIAL

***Proyecto Webots***



Raúl Jiménez García

Carlos Eduardo Perez Lopez

Aileen Sandoval González

***Índice***

* Objetivo
* Procedimiento
* Código
* Conclusiones

### ***Objetivo***

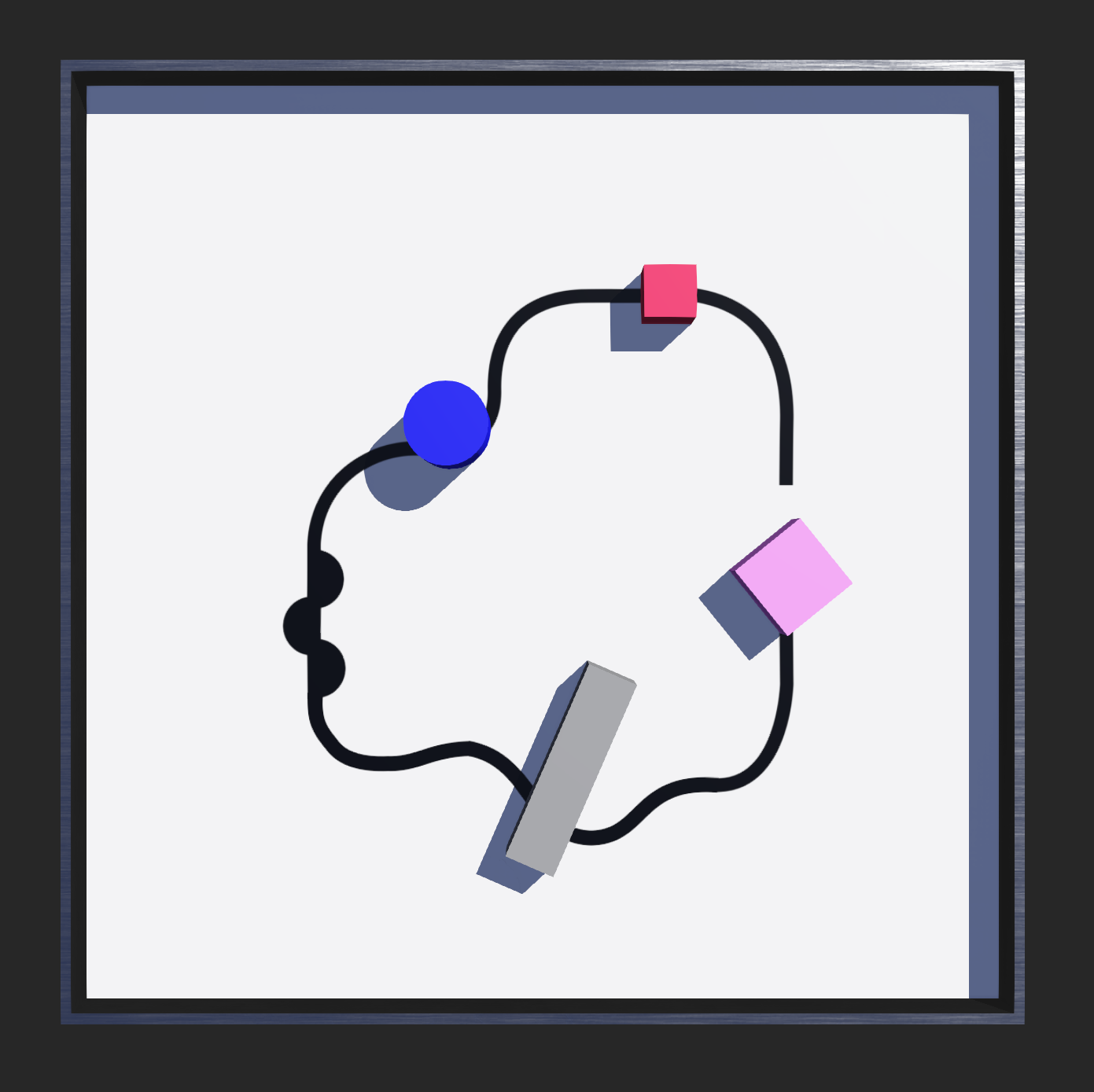
* Aprender a usar Webots y programar el funcionamiento de un robot.
* Nosotros decidimos programar un robot seguidor de línea negra que logre esquivar los obstáculos presentes y regrese al camino marcado.

### ***Procedimiento***

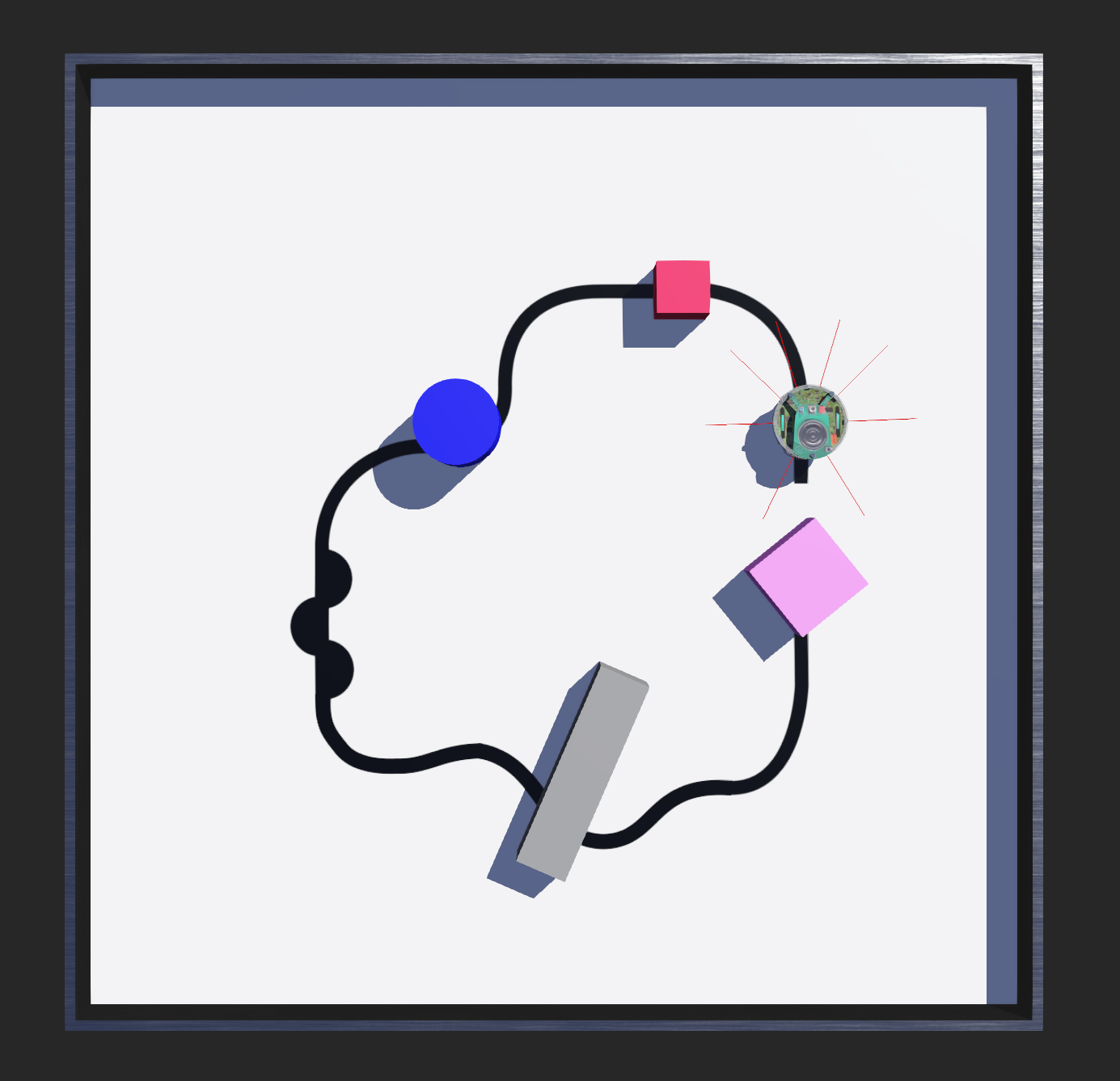
* *Desarrollo de la Arena*
  + Primero agregamos un Rectangle Arena a nuestro mundo.
  + Después modificamos los siguientes campos dentro de la arena:
    - floor Size: 0.9 0.9
    - floor Tile Size: 0.9 0.9
  + Agregamos una apariencia (PBR Appearance) a nuestra arena y le agregamos una imagen (Image Texture) en el campo base ColorMap.
    - En el campo url es donde vamos a agregar la imagen de la línea negra.



* *Desarrollo de los Obstáculos*
  + Vamos a agregar 4 objetos sólidos (Solid) a nuestro mundo.
  + En el campo de children vamos a agregar una forma (Shape) y posteriormente una apariencia (PBR Appearance) y geometría (Box o Cylinder).
  + Apariencia (PBR Appearance):
    - baseColor: en este caso rojo, azul, gris y rosa
    - roughness 0.5
    - metales 0
  + Geometría: para este proyecto utilizamos dos tipos de obstáculos, cajas y cilindros, y solo modificamos los tamaños y la orientación de estos objetos.
  + En el campo de bounding Object solo agregamos la geometría correspondiente de cada obstáculo.



* *Implementación del Robot E-Puck y sus modificaciones*
  + Agregamos un robot E-puck a nuestro mundo.
  + Después modificamos los siguientes campos del robot:
    - Colocamos al robot sobre la línea negra (translation y rotation).
    - controller: seleccionamos el archivo con el código del robot.
    - ground Sensors Slot: agregamos los sensores (E-puckGroundSensors).



* *Control del robot y sus módulos*
  + Buscamos el código del E-puck correspondiente al seguidor de línea negra.
  + Este código incluye los siguientes módulos:
    - Seguidor de línea (line following module)
      * Este módulo está creado bajo el concepto de los vehículos de Braitenberg. Tiene sensores primitivos (midiendo algún estímulo en un punto) y ruedas (cada una dirigida por su propio motor) que funcionan como actuadores o efectores. El sensor está directamente conectado a un efector, de modo que una señal percibida produce inmediatamente un movimiento de la rueda. Esto quiere decir que parecen esforzarse por alcanzar determinadas situaciones y evitar otras, cambiando de rumbo cuando la situación cambia.
    - Esquivar obstáculos (obstacle avoidance)
      * Este módulo, el E-puck detecta si hay un obstáculo en su camino. Este guarda la información de en qué lado se encuentra el obstáculo y lo esquiva dando la vuelta. Este módulo se activa si se encuentra un obstáculo frente al robot y se desactiva cuando no hay nada que le impida el movimiento.
    - Seguir obstáculo (obstacle following)
      * Esta función se encarga de hacer que el E-puck rodeé al obstáculo que se encuentra en su camino. Aquí el robot tiene una tendencia de girar hacia el lado donde se encuentra el objeto hasta que logra darle toda la vuelta.
    - Regreso a la línea (line entering)
      * Este módulo simplemente se encarga de regresar al E-puck a la línea negra cuando este previamente esquivó un obstáculo que se encontraba en su camino.
    - Main

.

### ***Código del Robot***

/\*

\* Copyright 1996-2020 Cyberbotics Ltd.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

#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];

//------------------------------------------------------------------------------

//

// BEHAVIORAL MODULES

//

//------------------------------------------------------------------------------

////////////////////////////////////////////

// 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].

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;

}

////////////////////////////////////////////

// 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].

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;

}

}

////////////////////////////////////////////

// LLM - Line Leaving Module

//

// Since it has no output, this routine is not completely finished. 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).

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 \*

}

}

}

}

////////////////////////////////////////////

// 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].

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;

}

}

////////////////////////////////////////////

// LEM - Line Entering Module

//

// This is the most complex module (and you might find easier to re-program it

// by yourself instead of trying to understand it ;-). 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 (i) the two lateral ground sensors, (ii) the argument "side"

// which determines the direction that the robot has to follow when detecting

// a black line, and (iii) the variable "lem\_reset" that resets the state to

// standby. The output speeds are stored in lem\_speed[LEFT] and

// lem\_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;

}

}

//------------------------------------------------------------------------------

//

// CONTROLLER

//

//------------------------------------------------------------------------------

////////////////////////////////////////////

// Main

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;

}

### ***Conclusión***

* Este proyecto estuvo fácil de realizar después de conocer más acerca de las simulaciones de Webots. El programa fue algo nuevo para nosotros porque nunca lo habíamos utilizado para simular y programar el comportamiento de un robot. Lo más difícil fue a la hora de programar el control del E-puck, ya que el código es muy complejo y sin ayuda de los ejemplos y las simulaciones esto hubiera sido imposible. El programa cuenta con varios ejemplos muy completos que te ayudan bastante a entender el uso de los robots.