UNIVERSITY
OF WOLLONGONG
IN DUBAI

Assignment Cover Sheet

Student Name	Student number	Student submitting work
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	

Subject code and name	CSCI291 – Programming for Engineers
Subject Coordinator	Dr. Abdsamad Benkrid
Tutorial Coordinator	Mr. Kiyan Afsari
Title of Assignment	CSCI291 Webots Project Report
Date and time due	[redacted]
Group Number	[redacted]
Total number of pages	[redacted]

Webots Project Report

CSCI291 Project Report

CSCI291 – Programming for Engineers

[redacted]

[redacted]

University of Wollongong in Dubai
School of Engineering

Aim

The aim of this project is to develop a Webots mobile robot application capable of solving a maze to locate the position with the maximum amount of light. The robot should navigate through the maze, identify all dead-end spots where light sources are located, and determine the light source with the highest intensity. Once identified, the robot should navigate directly to this brightest light source and come to a complete stop. The implementation should be done in C language, using various sensors such as light sensors and distance sensors, and should not be hardcoded for a specific map.

Initialization of Libraries and Variables

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <webots/camera.h>
#include <webots/distance_sensor.h>
#include <webots/motor.h>
#include <webots/robot.h>
#include <webots/position_sensor.h>
#include <webots/gps.h>
#include <webots/light_sensor.h>
#define MAX_SENSOR_NUMBER 16
...
static Position path[MAX_PATH_LENGTH];
static int path_length = 0;
static int junction_count = 0;
static int current_direction = 0; // 0: forward, 1: left, 2: right, 3: backward

static WbDeviceTag light_sensor;
static FILE *data_log;
```

The code initializes essential libraries to enable robot functionality in Webots simulation. Standard libraries like `<stdio.h>`, `<stdlib.h>`, `<string.h>`, and `<math.h>` provide basic operations including I/O, memory management, string manipulation, and mathematical calculations. Webots-specific libraries manage hardware interactions through distance sensors, motors, position sensors, GPS, and light sensors.

Constants and macros regulate sensor behaviour and navigation. `MAX_SENSOR_NUMBER(16)` defines maximum sensors, `RANGE` sets operational parameters, and `BOUND(x, a, b)` restricts sensor readings within limits. Navigation thresholds include `JUNCTION_THRESHOLD` for intersection detection, and `GPS_THRESHOLD` and `LIGHT_THRESHOLD` for position and light sensing. The code maintains motor controls, sensor references, and position tracking mechanisms through variables like `left_motor`, `right_motor`, and position sensors.

Data management utilizes a `matrix[MAX_SENSOR_NUMBER]` array for sensor readings, with `num_sensors` tracking active sensors and range defining operational boundaries. Time and speed controls include `time_step`, `max_speed`, and `speed_unit`. Position tracking employs `start_position`, `start_time`, and `has_moved` variables, with `CHECK_START_DELAY` set to 60,000 milliseconds.

Path navigation uses the path array limited by `MAX_PATH_LENGTH`, with `path_length` counting steps and `junction_count` tracking intersections. The `current_direction` variable monitors orientation (0-3 for forward, left, right, backward). Additional components include `light_sensor` reference and `data_log` pointer for debugging and analysis purposes.

Initialization Function

```
static void initialize() {
    wb_robot_init();
    ...
    printf("The %s robot is initialized, it uses %d distance sensors and GPS\n", robot_name,
num_sensors);
}
```

The initialize function sets up the robot's environment and components through the Webots API. It starts by calling `wb_robot_init()` and setting the `time_step` variable from `wb_robot_get_basic_time_step()`. The robot's name and sensor configurations are obtained, with specific handling for the e-puck robot type.

For e-puck robots, the function configures speed settings, initializes 8 sensors, and assigns calibration matrix values. The code processes each sensor through a loop, enabling them with the specified time step and setting up their calibration data. Non-e-puck robots trigger an error message and program termination.

Motor initialization includes setting up left and right wheel motors with infinite rotation capability and zero initial velocity. Position sensors for both wheels are enabled to track movement. Multiple light sensors are initialized in a loop, each enabled with the specified time step.

The final setup phase includes GPS device initialization, creation of a data log file for sensor and GPS data (with CSV headers) and recording of the simulation start time. The function concludes by printing an initialization confirmation message with the robot's name and sensor count. This comprehensive setup ensures all components are ready for operation in the simulated environment.

Update Path Memory Function

```
static void update_path_memory() {
    double left_pos = wb_position_sensor_get_value(left_position_sensor);
    ...
}
```

The `update_path_memory` function records the robot's path by storing position data in memory, essential for navigation and maze-solving tasks. It begins by retrieving current wheel positions through position sensors for both left and right wheels.

The robot's current position is calculated using wheel positions and trigonometric functions. X-coordinates are computed using the average wheel position multiplied by the cosine of the current direction (in radians), while y-coordinates use sine calculations. If the path length hasn't reached `MAX_PATH_LENGTH`, these coordinates are stored in the path array, and `path_length` is incremented.

This continuous position tracking enables the robot to maintain a record of its movements, facilitating features like backtracking and path analysis in maze navigation scenarios.

Detect Junction Function

```
static int detect_junction(double *sensors_value) {
    int left_open = 0, right_open = 0, front_open = 1;
    ...
    return (left_open + right_open + front_open) >= 2;
}
```

The `detect_junction` function determines junction points using the robot's distance sensors. It initializes three variables: `left_open` and `right_open` (set to 0), and `front_open` (set to 1), representing path availability in each direction.

The function processes each sensor's reading by normalizing values (subtracting sensor value/range from 1.0) to indicate obstacle proximity. Based on sensor position, it updates the corresponding direction variables: sensors in the first third indicate right paths, last third indicate left paths, and middle section represents forward paths. The normalization process helps identify clear paths when values exceed `JUNCTION_THRESHOLD`.

The function concludes by returning a Boolean value indicating a junction's presence, defined as having at least two open paths (sum of `left_open`, `right_open`, and `front_open` ≥ 2). This detection mechanism enables informed navigation decisions in maze environments.

Analyze Junction Function

```
static void analyze_junction(double *sensors_value) {
    int left_open = 0, right_open = 0, front_open = 1;
    ...
    junction_count++;
}
```

The `analyze_junction` function evaluates surroundings at junctions using distance sensor data to identify open paths and determine movement direction. Based on available paths and current orientation, it follows a decision hierarchy: turn left when available and not coming from right, turn right when available and not coming from left, continue straight if possible, or turn back as a last resort. The function tracks encountered junctions through `junction_count`, enabling effective maze navigation through sensor-based decision-making.

Is Back at Start Function

```
static int is_back_at_start() {
    const double *gps_values = wb_gps_get_values(gps);
    ...
    return distance < GPS_THRESHOLD;
}
```

The `is_back_at_start` function determines if the robot has returned to its starting position using GPS sensor data. It retrieves current GPS coordinates and calculates coordinate differences (`dx`, `dy`) between current and starting positions. The function then computes the distance using the formula $\sqrt{dx^2 + dy^2}$.
A debug statement prints coordinate differences, calculated distance, and `GPS_THRESHOLD` for monitoring purposes. The function returns true (1) if the calculated distance is less than `GPS_THRESHOLD`, and false (0) otherwise, enabling detection of successful return to initial position for navigation tasks like loop completion or home base return.

At Brightest Spot Function

```
static int at_brightest_spot() {
    const double *gps_values = wb_gps_get_values(gps);
    ...
    return distance < LIGHT_THRESHOLD;
}
```

The `at_brightest_spot` function determines if the robot has reached the maximum light intensity location using GPS data. It retrieves current coordinates and calculates differences (dx, dy) between current position and maximum light spot location. The Euclidean distance is computed using $\sqrt{dx^2 + dy^2}$.

Debug information includes coordinate differences, calculated distance, and LIGHT_THRESHOLD for monitoring. The function returns true (1) if the distance is less than LIGHT_THRESHOLD, and false (0) otherwise, enabling the robot to identify when it has reached the brightest location and respond accordingly.

Record Start Position Function

```c
static void record_start_position() {
    const double *gps_values = wb_gps_get_values(gps);
    ...
    printf("Start position recorded: (%.2f, %.2f)\n", start_position.x, start_position.y);
}
```

The `record_start_position` function captures the robot's initial position using GPS sensor data. It retrieves current coordinates through `wb_gps_get_values(gps)` and stores them in the start_position structure (`start_position.x` and `start_position.y`). The function confirms successful recording by printing the stored coordinates, enabling future navigation tasks like return-to-base operations.

Get Average Light Value Function

```c
double get_average_light_value() {
    double total_light = 0.0;
    ...
    return total_light / NUM_LIGHT_SENSORS;
}
```

The `get_average_light_value` function calculates the average light intensity detected by the robot's light sensors. It starts by initializing a variable `total_light` to 0.0. The function then iterates over each light sensor (assuming `NUM_LIGHT_SENSORS` is defined elsewhere in the code) and retrieves the light intensity value using `wb_light_sensor_get_value(light_sensors[i])`.

These values are summed up in `total_light`. After the loop, the function returns the average light intensity by dividing `total_light` by `NUM_LIGHT_SENSORS`. This function provides a single value representing the overall light intensity around the robot.

Read Sensor Function

```
void read_sensor() {
    double average_light = get_average_light_value();
    ...
    // Print to console
    ...
}
```

The `read_sensor` function processes sensor data, manages brightest spot tracking, and handles data logging. It collects average light intensity, GPS coordinates, and simulation time. When a higher light intensity is detected, it updates `max_average_light` and corresponding GPS coordinates (`brightest_spot_x`, `brightest_spot_y`, `brightest_spot_z`), printing a notification message.
The function records time, light intensity, and GPS coordinates in both a CSV file and console output, enabling both post-simulation analysis and real-time monitoring of the robot's status.

Delay Function

```
void delay(int milliseconds) {
    int start_time = wb_robot_get_time() * 1000;  // Get current simulation time in ms
    ...
}
```

The delay function pauses robot operations for a specified duration in milliseconds. It calculates start_time by converting current simulation time to milliseconds (`wb_robot_get_time() * 1000`). Using a while loop with `wb_robot_step` function, it continuously checks if the elapsed time (`current_time - start_time`) meets the specified delay. This timing mechanism enables precise control of robot operations and sensor reading synchronization.

Main Function

```
int main() {
    initialize();
    ...
    wb_robot_cleanup();
    return 0;
}
```

The main function manages the robot's complete operation cycle. It begins with initialization and sets `start_check_time` for position verification. The primary loop runs continuously, initializing speed arrays and sensor values. During the initial `START_DELAY` period, it records the start position and sets `has_moved` flag.

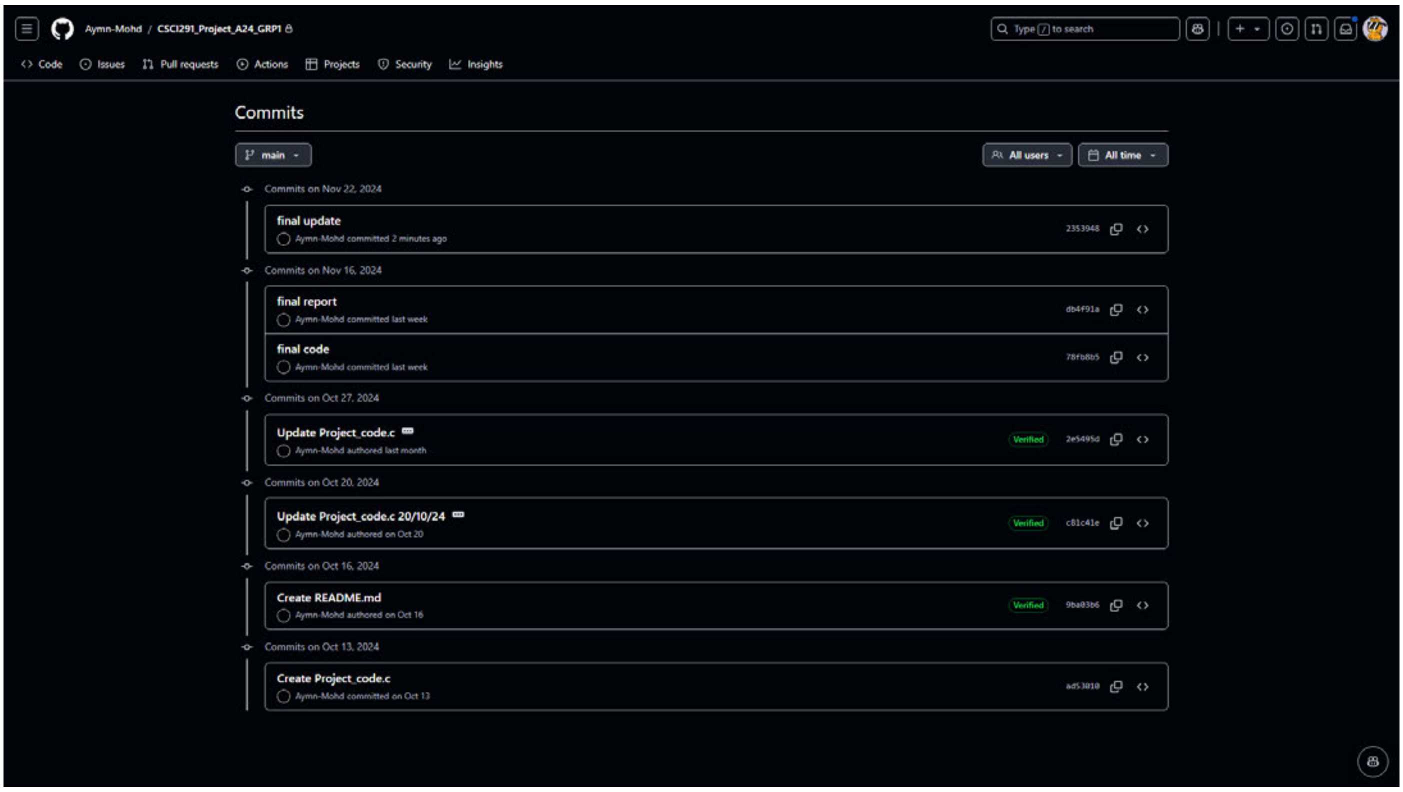
The function processes sensor data, updates path memory, and logs information. At junctions, it analyses surroundings to determine direction. Motor speeds are calculated using a modified Braitenberg algorithm and bounded within limits. After `start_check_time`, it verifies return to start position, stopping if confirmed.

Post-primary loop, it records maximum light data and enters a secondary loop focused on reaching the maximum light spot. The function concludes with playing a buzzer sound to indicate that it has reached the brightest spot in the maze, and initiates cleanup operations through `wb_robot_cleanup()`. This comprehensive control system enables efficient navigation, junction handling, and light-source targeting.

References

Cyberbotics Documentation: https://cyberbotics.com/doc/guide/foreword

GitHub Commit History



Commits

Commits on Nov 22, 2024

- **final update** — Aymn-Mohd committed 2 minutes ago — 2353948

Commits on Nov 16, 2024

- **final report** — Aymn-Mohd committed last week — db4f91a
- **final code** — Aymn-Mohd committed last week — 78fb8b5

Commits on Oct 27, 2024

- **Update Project_code.c** — Aymn-Mohd authored last month — Verified — 2e5495d

Commits on Oct 20, 2024

- **Update Project_code.c 20/10/24** — Aymn-Mohd authored on Oct 20 — Verified — c81c41e

Commits on Oct 16, 2024

- **Create README.md** — Aymn-Mohd authored on Oct 16 — Verified — 9ba03b6

Commits on Oct 13, 2024

- **Create Project_code.c** — Aymn-Mohd committed on Oct 13 — ad53010

[redacted]

GitHub Repo: https://github.com/Aymn-Mohd/CSCI291_Project_A24_GRP1

Project Video Link: https://youtu.be/FvJz4LkhChY