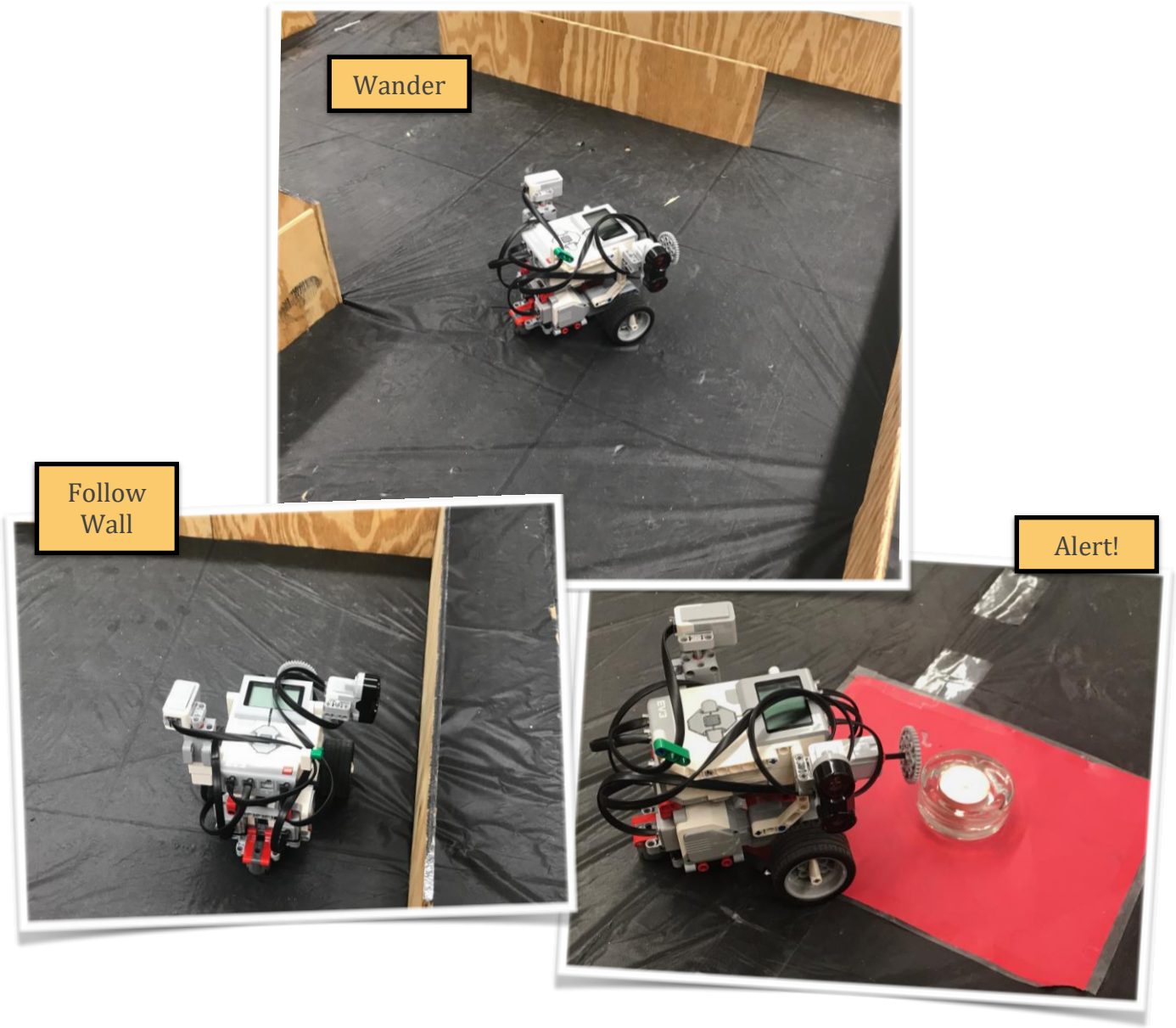
Team: Mindstorms
4 December 2017

Behaviour-based Fire Alarm Robot

To move from an unknown position in an in-door environment to look for a fire and raise an alarm



Fire Alarm Robot

Wander, Follow, Alert!

Setup

The environment is a closed in-door space separated by randomly placed walls. The robot is arbitrary put at an unknown location with unknown orientation with the world.

The Project

Robot Design

The robot uses a light intensity sensor, a motion sensor, and a sonar. The sonar is placed on the right side of the robot structure which is used to follow a wall on its right. The light intensity sensor is on the other side of the robot which scans a nearby environment for a possible fire source. The motion sensor is used in cases where the robot may hit the wall in front of it.

Behaviour Strategy

Our robot starts by checking if there is a nearby wall on its right. If found, it starts following the wall. If not, it starts to wander nearby. The wander algorithm is randomised to choose from three actions: turn left, turn right, go straight. The “wander” keeps running till it moves away from the current wall and finds another wall (it could happen that it finds the same wall again).

When the robot is following the wall, it “tries” to maintain an average distance range from wall (5cm - 6cm) If it gets too near or too far from the wall, it adjusts its speed accordingly and comes back in the acceptable distance range.

While it follows the wall, it may bump into another wall at a corner. This is detected by the motion sensor. The motion sensor behaviour is marked as high priority and the robot immediately turns left in order to follow the new wall found.

After a while of following the wall, to be precise for 20 seconds, the wander algorithm is run again. The wander algorithm runs for at least 5 seconds in order to make sure that it is away from the current wall that it followed.

Simultaneously, at all times, in a periodic interval of 6 seconds, the room intensity sensor scans the nearby environment for a possible fire source/candle. This is done by checking the initial room intensity (before the robot first started moving) and the currently scanned maximum intensity. If the difference in the intensities is more than a certain threshold, the robot know that there is a fire (in our case of candle, threshold being 6).

Challenges Faced

1. Make a “U-turn”: it may sometimes so happen that the wall ends abruptly. In such a case, the robot keeps turning sharp right in order to maintain the wall follow behaviour.

2. Maintaining the mean distance from the wall while following it. Based on the “how much near” or “how far” the robot it, the speed is varied to make sure that the robot does collide into the wall or move too far away from the wall respectively.

Final action

Once, the fire is detected, the robot additionally detects how far it is away from the candle. It then traverses in the direction of fire for that distance.

CODE

```
//============================================================================
=======

// Name        : WallFollower.cpp

// Author      : Mindstorms

// Description : Behaviour based autonomous robot for fire alarm

//============================================================================
=======



#include <ev3.h>

#include <ev3_button.h>

#include <ev3_command.h>

#include <ev3_constants.h>
```

```
#include <ev3_lcd.h>

#include <ev3_output.h>

#include <ev3sensor.h>

#include <cstdlib>

#include <ctime>

#include <string>


#define MIN_WALL_DIST 50

#define MAX_WALL_DIST 60

#define MEAN_WALL_DIST (MAX_WALL_DIST + MIN_WALL_DIST) / 2


#define SPEED 20 // CHANGE IN SPEED NEEDS UPDATE TO REVERSE LOGIC

#define DEVIATION_FACTOR 2


#define K_V 0.08

#define TURN_ANGLE 183


#define CIRCUMFERENCE 17.5929


#define INTENSITY_THRESHOLD 10


int speed_offset_b = 0, speed_offset_d = 0, deviation, wander_counter = 0,

                choice;
```

```
int room_intensity, currDistance, lastDistance, fireDistance;


enum Actions {
	WANDER, FIND_WALL, FOLLOW_WALL, MOVE_TO_WALL,
MOVE_AWAY_FROM_WALL
};
Actions currAction;


void turnLeft() {
	int currentCountD = MotorRotationCount(OUT_D);
	int currentCountB = MotorRotationCount(OUT_B);
	while (((TURN_ANGLE - (MotorRotationCount(OUT_D) - currentCountD)) > 10)
			&& ((TURN_ANGLE + (MotorRotationCount(OUT_B) -
currentCountB)) > 10)) {

		OnFwdReg(OUT_D,
				K_V*(TURN_ANGLE-(MotorRotationCount(OUT_D) -
currentCountD)) + 3);
		OnRevReg(OUT_B,
				K_V*(TURN_ANGLE+(MotorRotationCount(OUT_B) -
currentCountB)) + 3);
		Wait(100);
	}
	RotateMotor(OUT_D, 5,
			(TURN_ANGLE-(MotorRotationCount(OUT_D) - currentCountD)));
```

```
	RotateMotor(OUT_B, 5,
			(TURN_ANGLE+(MotorRotationCount(OUT_B) - currentCountB)) +
3);
}

void turnRight() {
	int currentCountD = MotorRotationCount(OUT_D);
	int currentCountB = MotorRotationCount(OUT_B);
	while (((TURN_ANGLE + (MotorRotationCount(OUT_D) - currentCountD)) > 10)
			&& ((TURN_ANGLE - (MotorRotationCount(OUT_B) - currentCountB))
> 10)) {

		OnFwdReg(OUT_B,
				K_V*(TURN_ANGLE-(MotorRotationCount(OUT_B) -
currentCountB)) + 3);
		OnRevReg(OUT_D,
				K_V*(TURN_ANGLE+(MotorRotationCount(OUT_D) -
currentCountD)) + 3);
		Wait(100);
	}
	RotateMotor(OUT_B, 5,
			(TURN_ANGLE-(MotorRotationCount(OUT_B) - currentCountB)));
	RotateMotor(OUT_D, 5,
			(TURN_ANGLE+(MotorRotationCount(OUT_D) - currentCountD)) +
3);
```

```
}

void moveForward(int distance) {
	distance /= 10;
	int FWD_ANGLE = distance / CIRCUMFERENCE * 360;

	int initialCountD = MotorRotationCount(OUT_D);
	int initialCountB = MotorRotationCount(OUT_B);

	while (((FWD_ANGLE - (MotorRotationCount(OUT_D) - initialCountD)) > 10)
			&& ((FWD_ANGLE - (MotorRotationCount(OUT_B) - initialCountB)) >
10)
			&& ButtonIsUp(BTNCENTER)) {

		OnFwdSync(OUT_BD,
				0.04*(FWD_ANGLE-abs(MotorRotationCount(OUT_D) -
initialCountD)) + 3);
		Wait(100);
	}
	RotateMotor(OUT_D, 5,
			(FWD_ANGLE-abs(MotorRotationCount(OUT_D) - initialCountD))-5);
	RotateMotor(OUT_B, 5,
			(FWD_ANGLE-abs(MotorRotationCount(OUT_B) - initialCountB))-5);
}
```

```
int getDistance() {
	return readSensor(IN_4) - 40;
}

bool scanEnvironment(int room_intensity) {
	int maxIntensity = 0, currIntensity = 0;
	int lSourceAngle = 0;

	while (ButtonIsUp(BTNCENTER) && MotorRotationCount(OUT_A) > -180) {
		currIntensity = readSensor(IN_1);
		OnRevReg(OUT_A, 5);
		if (currIntensity > maxIntensity) {
			maxIntensity = currIntensity;
			lSourceAngle = MotorRotationCount(OUT_A);
		}
		Wait(100);
	}
	while (ButtonIsUp(BTNCENTER) && MotorRotationCount(OUT_A) < 0) {
		OnFwdReg(OUT_A, 5);
		Wait(100);
	}
	Off(OUT_A);
```

```
        LcdPrintf(1, "%d %d ", maxIntensity, room_intensity);

        if (maxIntensity >= room_intensity + 6) {

                int currentCountD = MotorRotationCount(OUT_D);
                int currentCountB = MotorRotationCount(OUT_B);

                lSourceAngle = 2 * abs(lSourceAngle) + 10;

                while (((lSourceAngle - (MotorRotationCount(OUT_D) - currentCountD))
                                > 10)
                                && ((lSourceAngle + (MotorRotationCount(OUT_B) -
currentCountB))
                                                > 10) && ButtonIsUp(BTNCENTER)) {

                        OnFwdReg(OUT_D,
                                        K_V*(lSourceAngle-(MotorRotationCount(OUT_D) -
currentCountD)) + 3);
                        OnRevReg(OUT_B,
                                        K_V*(lSourceAngle+(MotorRotationCount(OUT_B) -
currentCountB)) + 3);
                        Wait(100);
                }
```

```
                RotateMotor(OUT_D, 5,
                                (lSourceAngle-(MotorRotationCount(OUT_D) -
currentCountD)));
                RotateMotor(OUT_B, 5,
                                (lSourceAngle+(MotorRotationCount(OUT_B) -
currentCountB)) + 3);

                turnLeft();
                int distance = getDistance();
                turnRight();

                moveForward(distance);
                return true;

        } else {
                return false;
        }
}

void wander() {
        currAction = WANDER;
        if (wander_counter++ == 50) {
                choice = rand() % 3;
                wander_counter = 0;
```

```
		}
		switch (choice) {
		case 0:
			OnFwdSync(OUT_BD, SPEED);
			break;
		case 1:
			turnLeft();
			wander_counter = 50;
			break;
		case 2:
			turnRight();
			wander_counter = 50;
			break;
		}

//		OnFwdSync(OUT_BD, SPEED);
}

void touchSensorActive() {
		OnRevSync(OUT_BD, SPEED-10);
		Wait(900);
		turnLeft();
}
```

```
void findWall() {
	currAction = FIND_WALL;
	OnFwdReg(OUT_B, SPEED + 40);
	OnFwdReg(OUT_D, SPEED);
}

void followWallStraight() {
	currAction = FOLLOW_WALL;
	if (currDistance > MEAN_WALL_DIST) {
		speed_offset_b = currDistance - MEAN_WALL_DIST;
	} else {
		speed_offset_d = MEAN_WALL_DIST - currDistance;
	}
	if (lastDistance < currDistance) {
		OnFwdReg(OUT_B, SPEED + (DEVIATION_FACTOR * speed_offset_d));
		OnFwdReg(OUT_D, SPEED + (DEVIATION_FACTOR * speed_offset_b));
	} else {
		OnFwdReg(OUT_B, SPEED + (DEVIATION_FACTOR * speed_offset_b));
		OnFwdReg(OUT_D, SPEED + (DEVIATION_FACTOR * speed_offset_d));
	}
	speed_offset_b = 0;
	speed_offset_d = 0;
```

```
}

void moveAwayFromWall() {
	currAction = MOVE_AWAY_FROM_WALL;
	deviation = MIN_WALL_DIST - currDistance;
	OnFwdReg(OUT_B, SPEED);
	OnFwdReg(OUT_D, SPEED + ((int) DEVIATION_FACTOR * deviation));
}

int moveTowardsWall() {
	currAction = MOVE_TO_WALL;
	deviation = currDistance - MAX_WALL_DIST;
	OnFwdReg(OUT_B, SPEED + ((int) DEVIATION_FACTOR * deviation));
	OnFwdReg(OUT_D, SPEED);
	return deviation;
}

void followWall() {
	int while_counter = 0, action_counter = 0, find_wall_action_counter = 0;

	while (ButtonIsUp(BTNCENTER)) {
		while_counter++;
		Off(OUT_BD);
```

```
//                LcdClean();
                currDistance = readSensor(IN_4);
                if (readSensor(IN_2) == 1) {
                        touchSensorActive();
                } else if (while_counter == 60) {
                        while_counter = 0;
                        if (scanEnvironment(room_intensity)) {
                                break;
                        } else {
                                continue;
                        }
                } else if (currDistance > 400
                                && (currAction != FIND_WALL || action_counter > 50)) {
                        find_wall_action_counter = 0;
                        action_counter = 0;
                        wander();
                } else if (currDistance > 80 && action_counter++ < 50) {
                        find_wall_action_counter = 0;
                        findWall();
                } else if (currDistance >= MIN_WALL_DIST && currDistance <
MAX_WALL_DIST) {
                        if (find_wall_action_counter++ > 200) {
                                wander();
                        } else {
```

```
                    action_counter = 0;
                    followWallStraight();
                }
            } else if (currDistance < MIN_WALL_DIST) {
                if (find_wall_action_counter++ > 200) {
                    wander();
                } else {
                    action_counter = 0;
                    moveAwayFromWall();
                }
            } else if (currDistance >= MAX_WALL_DIST) {
                if (find_wall_action_counter++ > 200) {
                    wander();
                } else {
                    action_counter = 0;
                    moveTowardsWall();
                }
            }
            lastDistance = currDistance;
            Wait(100);
        }
}
```

```
void init() {
	InitEV3();
	ResetRotationCount(OUT_D);
	ResetRotationCount(OUT_B);
	ResetRotationCount(OUT_A);
	setAllSensorMode(COL_AMBIENT, TOUCH_PRESS, NO_SEN, US_DIST_MM);
	srand(time(NULL));
	room_intensity = readSensor(IN_1);
}

int main() {
	init();

    followWall();

	ButtonWaitForPressAndRelease(BTNCENTER);

	FreeEV3();
}
```