CSE 4360

Team 4

Akshay Kumar Paluri , Daniel Thomas Palma, Edward Alkire, Hyun Ho Kim

Project 2

Intro

Our robot takes a relative priority approach. This allows for the robot to explore around the map until it finds a specific location by incorporating the touch, color, and sonic sensor. The top priority is always goal checking, but while the goal color has not been detected, the modes have different sub-priorities. In wall following mode, the priority is the touch sensors, then the sonar distance. In wandering mode, the priority is the touch sensors.

Wall following becomes priority based on its input from the sonic sensor, which is the distance from the wall. Wandering becomes priority when the robot receives a reading that is too far, which indicates that there is no wall there (so we can assume a new room has been found), and after a certain amount of time has passed since the first reading occurred and is still accurate.

Physical Design

As a team, we prioritized the design of the robot for maximum wall following accuracy. To accomplish this, the sonic sensor was placed at a 45-degree angle so that it could read wall distance in front of it while there was a wall to the robots left. For touch, we attached a bumper to the sensors and supported the weight using a bar. The reason for using a bar for support was because without it, the bumper would sag, jamming the sensors. We also chose to use two touch sensors so that we could distinguish what side the robot was hitting a wall with. For movement, we stayed with the same design, utilized two wheels with a steel ball in the back. To detect the fire, we chose to place the sensor close to the ground, approximately 1.5 cm, so that it could detect the colored tile accurately.

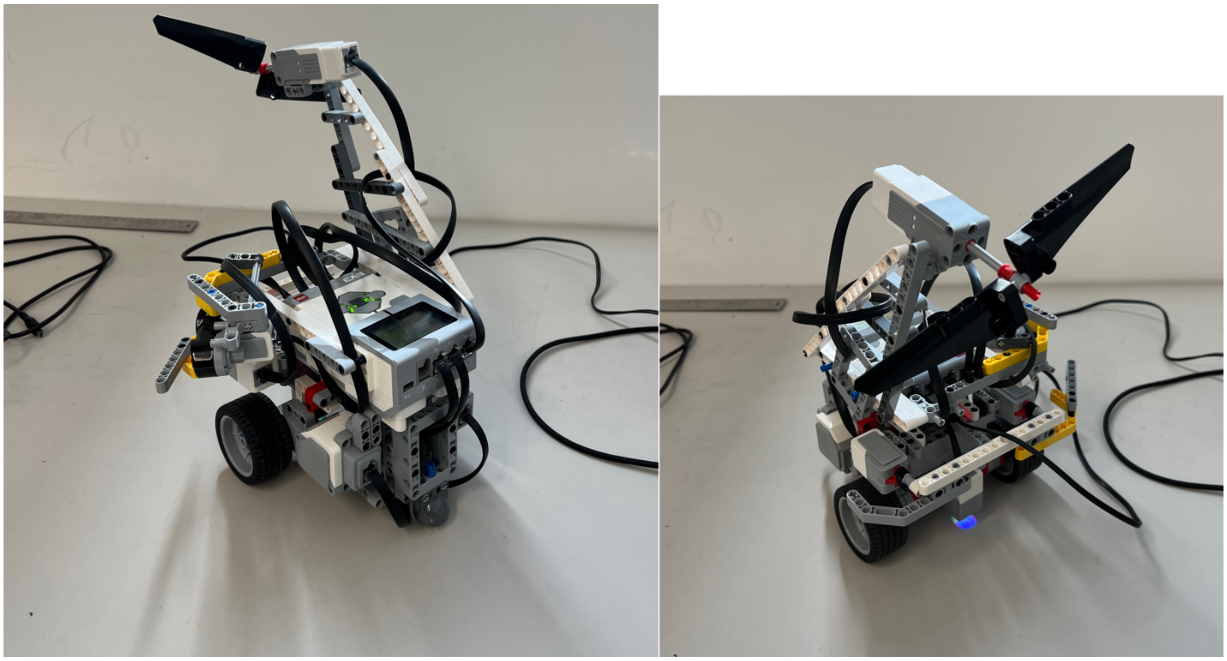

Fig 1: picture of robot

CSE 4360
Akshay Kumar Paluri , Daniel Thomas Palma, Edward Alkire, Hyun Ho Kim

Control System

The important components of the control system are as follows. wall() for wall following, true_wander() for wandering, and goal_check() for locating the fire as well as extinguishing it. The function wall() continuously reads the sonic and touch sensors. If the touch sensors get pressed, the function wander() is called and handles the logic for determining which way the robot should turn to prevent the robot from turning back into a corner.

The function true_wander() is called when the robot senses a distance greater than WALL_SENS_DIST and that distance remains above WALL_SENS_DIST for a pre-determined amount of time. It wanders by driving forward while checking the goal then, when it bumps into a wall, reverses, turns either left or right, then continues with that sequence for a predetermined mount of times. After it bumps the number of times determined, it drives straight, bumps into a wall, and then goes back into wall following mode.

The robot continuously checks for the goal using goal_check(). Once color sensor receives a color signal matching the goal area, it becomes the priority, then stops the wheels, and begins spinning the fan to extinguish the fire.

Experience

The experience with the system was as expected. In the beginning we over prioritized wall following to the point that it was a detriment to the objective of the robot. To fix this, we added a function that alternates between turning left or right during wandering. We also adjusted the function wander() so that it would call true_wander() whenever the wall was lost which allowed it to wander more often and explore more of the map. We also had to develop a method to determine whether the robot should turn in to a new area. The method chosen was to assume a missing wall led to a new area. This functionality was left to be simple so that it could be applied to any map given.

Instructions

To run program, load program on to robot designed similarly to ours with the sonic sensor at a 45-degree angle. Upload the file using the LEGO Mindstorms EV3 MicroPython VS Code extension. Next, run by going into file browser and selecting with center button on LEGO EV3.

thing.py

```
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.media.ev3dev import SoundFile, ImageFile
import time
import sys
import math
# This program requires LEGO EV3 MicroPython v2.0 or higher.
# Click "Open user guide" on the EV3 extension tab for more information.

MOVE_SPEED = 300
WALL_SENS_DIST = 150
WANDER_TURN_TIME = 825
WANDER_FWD_TIME = 900
WANDER_BACK_TIME = 900

LEFT = True
LEFT_COUNT = 0

# Create your objects here.
ev3 = EV3Brick()
lw = Motor(Port.C)
rw = Motor(Port.B)
lt = TouchSensor(Port.S4)
rt = TouchSensor(Port.S1)
cr = ColorSensor(Port.S2)
sonic = UltrasonicSensor(Port.S3)
wind = Motor(Port.A)

# Write your program here.

def true_wander(lw, rw, cr, sonic, speed, ev3, wind, lt, rt):
    global LEFT
    global LEFT_COUNT
    inner_bool = LEFT
    print(inner_bool, LEFT)
    bumpCount = 0
    while 1:
        while 1:
            # Drive until bump wall
            lw.run_time(speed, WANDER_FWD_TIME, then=Stop.HOLD, wait=False)
            rw.run_time(speed, WANDER_FWD_TIME, then=Stop.HOLD, wait=True)
            goal_check(cr, ev3, wind, lw, rw)
            if lt.pressed() or rt.pressed():
```

```python
                break
        # Drive backward away from wall
        lw.run_time(-speed, WANDER_BACK_TIME, then=Stop.HOLD, wait=False)
        rw.run_time(-speed, WANDER_BACK_TIME, then=Stop.HOLD, wait=True)
        goal_check(cr, ev3, wind, lw, rw)
        # Turn to left
        if LEFT == True:
            if bumpCount == 3:
                print("right")
                rw.run_time(-speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=False)
                lw.run_time(speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=True)
                goal_check(cr, ev3, wind, lw, rw)
            else:
                lw.run_time(-speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=False)
                rw.run_time(speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=True)
                goal_check(cr, ev3, wind, lw, rw)
        # Turn right on every other go of the loop
        elif LEFT == False:
            if bumpCount == 3:
                print("left")
                lw.run_time(-speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=False)
                rw.run_time(speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=True)
                goal_check(cr, ev3, wind, lw, rw)
            else:
                rw.run_time(-speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=False)
                lw.run_time(speed, WANDER_TURN_TIME, then=Stop.HOLD, wait=True)
                goal_check(cr, ev3, wind, lw, rw)
        # bumpCount increase and loop again until 10 bumps occur
        bumpCount += 1
        if bumpCount == 4:
            LEFT_COUNT += 1
            if LEFT == False:
                LEFT_COUNT += 1
            if LEFT_COUNT == 2:
                LEFT = not LEFT
                LEFT_COUNT = 0
            break
    # After 3 bumps, drive until wall is bumped
    while 1:
        lw.run(speed)
        rw.run(speed)
        goal_check(cr, ev3, wind, lw, rw)
        if lt.pressed() or rt.pressed():
            break
    # Drive backward away from wall
    lw.run_time(-speed, WANDER_BACK_TIME, then=Stop.HOLD, wait=False)
    rw.run_time(-speed, WANDER_BACK_TIME, then=Stop.HOLD, wait=True)
    goal_check(cr, ev3, wind, lw, rw)
    while 1:
        # Spin until wall is found, break back into wander at < WALL_SENS_DIST
        # Should go back directly into wall()
        rw.run_time(-speed, 450, then=Stop.HOLD, wait=False)
        lw.run_time(speed, 450, then=Stop.HOLD, wait=True)
        goal_check(cr, ev3, wind, lw, rw)
        if(sonic.distance() < WALL_SENS_DIST):
            break

def wander(lw, rw, cr, sonic, speed, ev3, wind, lt, rt):
    movement_selector = 1
    while(1):
        goal_check(cr, ev3, wind, lw, rw)
        if(lt.pressed()):
            # If lt pressed and distance > 360, turn left
            if(sonic.distance() > 360):
                rw.stop()
                lw.run_time(-speed, 1000, wait=True)
                if(sonic.distance() < WALL_SENS_DIST):
                    break
            else:
                lw.stop()
                rw.run_time(-speed,900, wait=True)
                if(sonic.distance() < WALL_SENS_DIST):
                    break
        elif(rt.pressed()):
            # If rt pressed and distance > 360, turn left
            if(sonic.distance() > 360):
                rw.stop()
                lw.run_time(-speed, 1000, wait=True)
                if(sonic.distance() < WALL_SENS_DIST):
                    break
            else:
                lw.stop()
                rw.run_time(-speed, 900, wait=True)
                if(sonic.distance() < WALL_SENS_DIST):
                    break
        else:
            # Do not delete. Alter value so that results are better
            # if distance < WALL_SENS_DIST, go back into wall
            # else, true_wander()
            lw.run_time(-speed, 0, wait=False)
            rw.run_time(speed, 0, wait=True)
            if(sonic.distance() < WALL_SENS_DIST):
                break
            else:
                goal_check(cr, ev3, wind, lw, rw)
                true_wander(lw, rw, cr, sonic, speed, ev3, wind, lt, rt)
                while 1:
                    # always return state of LEFT but turn if sonic distance too large
                    if(sonic.distance() > WALL_SENS_DIST):
                        lw.run_time(-speed, 300, then=Stop.HOLD, wait=False)
                        rw.run_time(speed, 300, then=Stop.HOLD, wait=True)
                    else:
                        break
                break

def goal_check(cr, ev3, wind, lw, rw):
    if(cr.color() == Color.WHITE or cr.color() == Color.BLUE or cr.color() == Color.YELLOW):
        lw.stop()
        rw.stop()
        wind.run_time(1000,10000,wait=True)
        sys.exit()

def wall(lw, rw, cr, sonic, speed, ev3, wind, lt, rt):
    start = time.time()
    time_flag = 0
    fflag = 0
    while(1):
        goal_check(cr, ev3, wind, lw, rw)
        if(lt.pressed() or rt.pressed()):
            wander(lw, rw, cr, sonic, speed, ev3, wind, lt, rt)

        else:
            goal_check(cr, ev3, wind, lw, rw)
            if(time_flag == 0 and (sonic.distance() >= WALL_SENS_DIST and sonic.distance() <= 3000)):
                start = time.time()
                time_flag = 1
            if(sonic.distance() >= WALL_SENS_DIST and sonic.distance() <= 3000):
                end = time.time()
                if(end-start >= 2.25):
                    # May need to be changed to true_wander
                    # true_wander(lw, rw, cr, sonic, speed, ev3, wind, lt, rt)
                    wander(lw, rw, cr, sonic, speed, ev3, wind, lt, rt)
                    start = 0
                    time_flag = 0
                else:
                    lw.run(speed/2.66)
                    rw.run(speed)
            if(sonic.distance() >= 95 and sonic.distance() <= WALL_SENS_DIST):
                time_flag = 0
                start = 0
                lw.run(speed/2.66)
                rw.run(speed)
            elif(sonic.distance() >= 65 and sonic.distance() < 95):
                time_flag = 0
                start = 0
                lw.run(speed)
                rw.run(speed)
            elif(sonic.distance() < 65):
                time_flag = 0
                start = 0
                lw.run(speed)
                rw.run(speed/2)

wall(lw, rw, cr, sonic, MOVE_SPEED, ev3, wind, lt, rt)

```