ECS 418: Intelligent Robotics
Assignment-2

Jaydev Singh Rao
19147
jaydev19@iiserb.ac.in

Jayesh Kumpawat
19148
jayesh19@iiserb.ac.in

12^{th} November, 2022

Contents

1 Problem Statement **1**

2 Solution **1**
2.1 Approach: 1 1
2.1.1 Robot Controller Implementation 1
2.1.2 Problems with Approach 1 2
2.2 Approach: 2 2
2.2.1 Environment 2
2.2.2 Robot Controller Implementation 3
2.2.3 Results 3

3 Appendix **3**
3.1 Supervisor for the environment 3
3.2 Controller for the robot 5

1 Problem Statement

Planning a path for a robot towards a goal location in the presence of some stationary and mobile obstacles.

1. The robot is given the location of the goal where it has to reach.
2. The robot has no knowledge about the environment and the obstacles.
3. The mobile obstacles move in a straight line with a constant velocity.

2 Solution

2.1 Approach: 1

2.1.1 Robot Controller Implementation

For the first approach we planned the robot navigation in three parts:

- **Move to Goal**: For this part we used the carrot chasing algorithm. In this, the robot tries to move along the line from start location to the goal location.

- **Avoiding Static Obstacles**: For avoiding the static obstacle we tried the *Bug algorithms* (both Bug-0 and Bug-2). The robot is simply supposed to follow the obstacle boundary till the condition of leave point is met.

- **Avoiding Moving Obstacles**: For avoiding mobile obstacles we planned on using a control based approach where the robot tried to keep the relative velocity positive.

2.1.2 Problems with Approach 1

The approach 1 failed for us because of following reasons:

1. The robot could not determine the required leave point on the boundary of non moving cylindrical obstacles for our implementation of *Bug algorithms*.

2. We were not able to implement the controller for keeping the relative velocity greater than 0 for moving obstacles with the amount of information given to the robot controller. Any such implementation would require more sensitive sensors and complicated controllers than feasible in *Webots*. Moreover, computing the relative velocity would require the distance between the robot and the obstacles, which is not possible without having the precise location of the obstacles.

2.2 Approach: 2

2.2.1 Environment

- The experiment was performed on a *Webots* environment. It consists of a rectangular arena comprising 8 stationary obstacles, 3 mobile obstacles, a start and an end location.

- **Mobile obstacle:** The 3 solid balls (obstacles) follow straight lines. This is implemented using a supervisor controller. (Section 3.1)

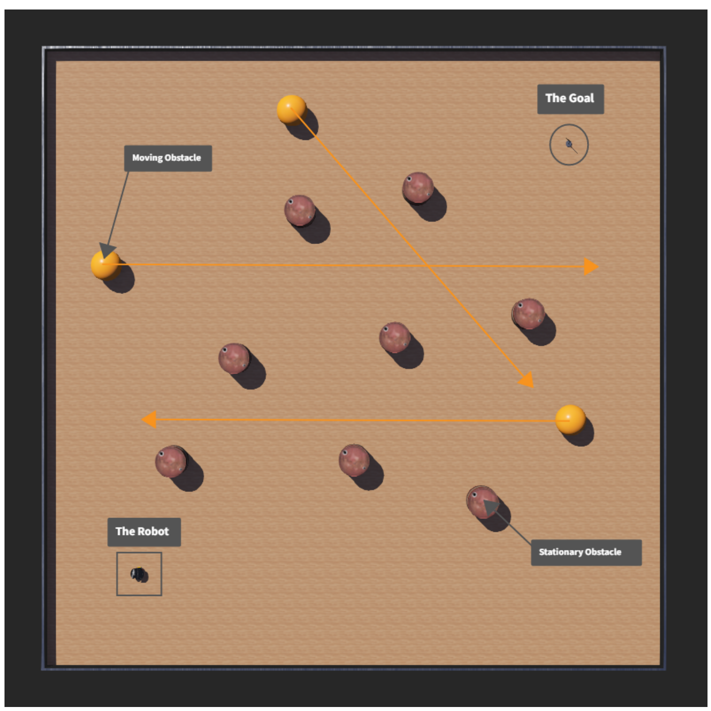

Figure 1: The Environment for the test simulation.

2.2.2 Robot Controller Implementation

Proportional controls are created for moving to the goal and avoiding obstacles:

- **Goal alignment:** The robot's path towards the goal is controlled proportionally by the difference between heading angle and the goal. (Line 97-98 Section 3.2)
- **Obstacle-avoidance:** Three distance sensors are mounted on the robot, which provides the distance between the robot and obstacles. Similar to goal alignment, proportional controls are created for the robot in terms of its distance from the obstacles. (Line 101-117 Section 3.2)

2.2.3 Results

The complete video of a test simulation can be found at `https://youtu.be/mERM_7T_vDg`. The screenshots are given below:

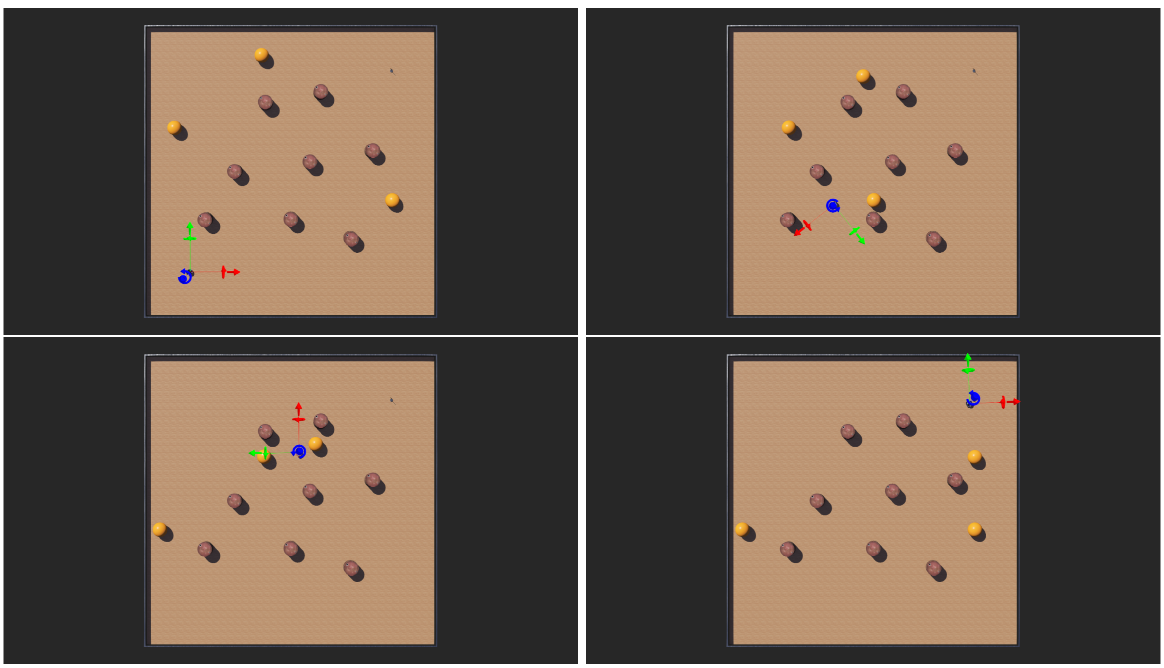

Figure 2: Screenshots of a test simulation

3 Appendix

3.1 Supervisor for the environment

(Github link)

```python
"""Supervisor controller."""

from controller import Supervisor
import math

def angle(robot_pos, goal_pos):
    x = robot_pos[0] - goal_pos[0]
    y = robot_pos[1] - goal_pos[1]
    rad = math.atan2(y, x)
    return rad + math.pi
```

```python

def dist_betn(p1, p2):
    return math.sqrt((p1[0] - p2[0]) ** 2 \
            + (p1[1] - p2[1]) ** 2)


robot = Supervisor()

############## Moving Obstacles ##############

ball_a = robot.getFromDef("BALL_A")
ball_b = robot.getFromDef("BALL_B")
ball_c = robot.getFromDef("BALL_C")

print(ball_a)

position_a = ball_a.getPosition()
position_b = ball_b.getPosition()
position_c = ball_c.getPosition()

tran_a = ball_a.getField('translation')
tran_b = ball_b.getField('translation')
tran_c = ball_c.getField('translation')

target_a = [-9, -2, 0.5]
target_b = [7, -2, 0.5]
target_c = [7, 3, 0.5]

velocity_a = dist_betn(target_a, position_a) / 1000
velocity_b = dist_betn(target_b, position_b) / 3000
velocity_c = dist_betn(target_c, position_c) / 3000


angle_a = angle(position_a, target_a)
angle_b = angle(position_b, target_b)
angle_c = angle(position_c, target_c)

timestep = 64

while robot.step(timestep) != -1:

    position_a = ball_a.getPosition()
    position_b = ball_b.getPosition()
    position_c = ball_c.getPosition()

    delta_pos_a = [velocity_a * math.cos(angle_a),
                   velocity_a * math.sin(angle_a), 0]

    delta_pos_b = [velocity_b * math.cos(angle_b),
                   velocity_b * math.sin(angle_b), 0]

    delta_pos_c = [velocity_c * math.cos(angle_c),
                     velocity_c * math.sin(angle_c), 0]

    if dist_betn(target_a, position_a) > 0.5:
        position_a = [position_a[i] + delta_pos_a[i]
                        for i in range(3)]
        tran_a.setSFVec3f(position_a)
    else:
        tran_a.setSFVec3f(target_a)

    if dist_betn(target_b, position_b) > 0.5:
        position_b = [position_b[i] + delta_pos_b[i]
                        for i in range(3)]
        tran_b.setSFVec3f(position_b)
```

```python
    else:
        tran_b.setSFVec3f(target_b)

    if dist_betn(target_c, position_c) > 0.5:
        position_c = [position_c[i] + delta_pos_c[i]
                        for i in range(3)]
        tran_c.setSFVec3f(position_c)
    else:
        tran_c.setSFVec3f(target_c)

############### Controller End ################
```

3.2 Controller for the robot

(Github link)

```python
from controller import Robot, Motor, DistanceSensor
import math

############# Helper Functions ################
def get_bearing_in_degrees(north):
    rad = math.atan2(north[0], north[1])
    bearing = (rad - 1.5708) / 3.14 * 180.0
    bearing += 180
    if bearing < 0.0:
        bearing = bearing + 360.0
    return bearing

def angle_of(A, B):
    return math.degrees(math.atan2((B[1]-A[1]), (B[0]-A[0])) % 360) + 90

def distance_between(p1, p2):
    return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)

def distance_from_line(x, A, B):
    length_AB = distance_between(A, B)
    distance_from_AB = abs((B[1]-A[1])*(A[0] - x[0]) - (B[0]-A[0])*(A[1]-x[1]))
    distance_from_AB /= length_AB
    return distance_from_AB

def on_line(x, A, B, tolerance=0.02):
    dist = distance_from_line(x, A, B)
    if dist > tolerance:
        return False
    else:
        return True

def clamp(x, min, max):
    if x < min:
        return min
    elif x > max:
        return max
    else:
        return x

############ Controller Begin ##############

robot = Robot()

# Constants
TIME_STEP = 64
MAX_SPEED = 15.0
GOAL_POSITION = [7.0, 7.0, 0.0]
```

```
POS_EPSILON = 0.4  # distance from goal when to stop

#initialize motors
left_motor = robot.getDevice('left wheel')
right_motor = robot.getDevice('right wheel')
left_motor.setPosition(float('inf'))  # number of radians the motor rotates
right_motor.setPosition(float('inf'))
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)

# initialize devices
sense_left = robot.getDevice('sense_left')
sense_right = robot.getDevice('sense_right')
sense_left.enable(TIME_STEP)
sense_right.enable(TIME_STEP)

sense_front = robot.getDevice('sense_front')
sense_front.enable(TIME_STEP)

gps = robot.getDevice('gps')
gps.enable(TIME_STEP)

compass = robot.getDevice('compass')
compass.enable(TIME_STEP)

state = 'moving'

current_position = gps.getValues()
goal_distance_start = distance_between(current_position, GOAL_POSITION)


while robot.step(TIME_STEP) != -1 and state != 'reached_goal':
    # read sensors outputs
    left_value = 20.24 * sense_left.getValue()**(-4.76) + 0.6632
    right_value = 20.24 * sense_right.getValue()**(-4.76) + 0.6632
    front_value = 20.24 * sense_front.getValue()**(-4.76) + 0.6632

    current_position = gps.getValues()
    current_angle = get_bearing_in_degrees(compass.getValues())

    # initialize motor speeds at 50% of MAX_SPEED.
    left_speed  = 0.5 * MAX_SPEED
    right_speed = 0.5 * MAX_SPEED

    goal_distance = distance_between(current_position, GOAL_POSITION)
    goal_angle = angle_of(current_position, GOAL_POSITION)
    angle_diff = goal_angle - current_angle

    # proportional control to turn towards goal
    left_speed -= 0.5 * MAX_SPEED * angle_diff / 180.0
    right_speed += 0.5 * MAX_SPEED * angle_diff / 180.0

    # closest bstacle to the right
    if right_value < 3.0 and right_value < left_value and right_value < front_value:
        left_speed -= 0.25 * MAX_SPEED * (3.0 - right_value) / 3.0
        right_speed += 0.25 * MAX_SPEED * (3.0 - right_value) / 3.0

    # closest obstacle to the left
    if left_value < 3.0 and left_value < right_value and left_value < front_value:
        left_speed += 0.25 * MAX_SPEED * (3.0 - left_value) / 3.0
        right_speed -= 0.25 * MAX_SPEED * (3.0 - right_value) / 3.0

    # if obstacle in front, turn away from it
    if front_value < 3.0:
        if left_value > right_value:
```

```
            left_speed = -0.5 * MAX_SPEED
            right_speed = 0.5 * MAX_SPEED
        else:
            left_speed = 0.5 * MAX_SPEED
            right_speed = -0.5 * MAX_SPEED

    # clamp the speed to the maximum allowed speed
    left_speed = clamp(left_speed, -MAX_SPEED, MAX_SPEED)
    right_speed = clamp(right_speed, -MAX_SPEED, MAX_SPEED)

    print(left_speed / MAX_SPEED)
    print(right_speed / MAX_SPEED)
    # If we are close to the goal, stop
    if distance_between(current_position, GOAL_POSITION) < POS_EPSILON:
        state = 'reached_goal'
        left_motor.setVelocity(0.0)
        right_motor.setVelocity(0.0)
    else:
        left_motor.setVelocity(left_speed)
        right_motor.setVelocity(right_speed)

############ Controller End ##############
```