به نام خدا

تمرین سری سوم درس رباتیکز

گروه ۱۲: افتخاری – برزوئی – غفارنیا

سوال ۱:

ابتدا به پیاده‌سازی الگوریتم Bug 1 میپردازیم. تابع‌های استفاده شده در کد این قسمت، همانند تابع‌های سوال ۳ هستند با این تفاوت که در تابع avoid_obstacle_velocities، تغییراتی ایجاد شده است:

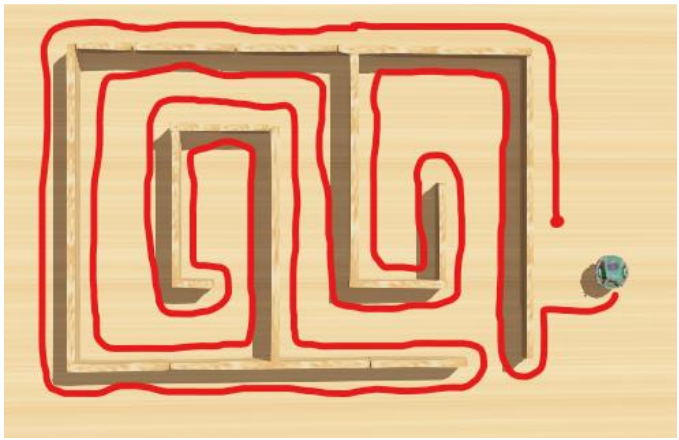
```
def avoid_obstacle_velocities():
    global turning_to_follow, follow_boundary, theta_at_start_of_turning
    global left_follow_dist, cnt, turning_in_the_end, HIT_POINT, chase_goal
    global change_hit_point_cnt
    obstacle = where_is_obstacle()

    cnt += 1
    if turning_to_follow:
        vl, vr = turn_left()
        theta = get_robot_heading(compass.getValues())
        if abs(theta - theta_at_start_of_turning) > 88:
            turning_to_follow = False
            follow_boundary = True
```

```
    elif follow_boundary:
        front, right, _, _, _ = get_distances()

        if front <= DIST_THRESHOLD:
            turning_to_follow = True
            theta_at_start_of_turning = get_robot_heading(compass.getValues())
            follow_boundary = False

        vl, vr = [FOLLOW_BOUNDARY_VELOCITY]*2

        vl -= K_F * (DIST_THRESHOLD - right)
        vr += K_F * (DIST_THRESHOLD - right)


        x, y = gps.getValues()[:2]

        if dist((x, y), HIT_POINT) < 10**(-2) and (cnt - change_hit_point_cnt > 1000):
            follow_boundary = False
            chase_goal = True

        if dist((x, y), (gx, gy)) < dist(HIT_POINT, (gx, gy)):
            HIT_POINT = (x, y)
            change_hit_point_cnt = cnt


    elif chase_goal:
        return get_velocities()
```

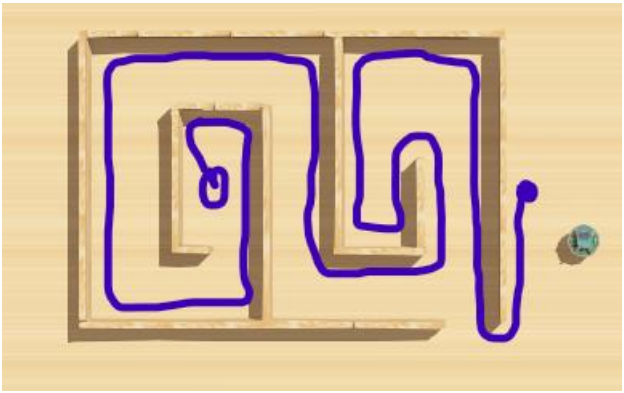
```
else:

    if obstacle is None:
        return get_velocities()

    elif obstacle == 'FRONT':
        vl, vr = handle_front_obstacle()

    elif obstacle == 'FRONT_LEFT':
        vl, vr = turn_right()

    elif obstacle == 'FRONT_RIGHT':
        vl, vr = turn_left()

    elif obstacle == 'LEFT':
        follow_boundary = True
        vl, vr = turn_right()

    elif obstacle == 'RIGHT':
        follow_boundary = True
        vl, vr = turn_left()

    else:
        return get_velocities()

return normalize(vl, vr)
```

الگوریتم Bug 1 بدین شکل عمل میکند که ابتدا از نقطه‌ی شروع، به سمت نقطه‌ی هدف حرکت میکند تا زمانی که به مانع برسد؛ سپس کل مانع را به طور کامل طی میکند تا نقشه محیط را بدست بیاورد و سپس بهترین مسیر برای رسیدن به نقطه مقصد را، تعیین میکند و به سمت آن حرکت کرده و در آن می‌ایستد.

مسیر حرکت ربات به شکل زیر است:



همانطور که توضیح داده شد و در شکل بالا قابل مشاهده است، ربات ابتدا کل موانع موجود را طی میکند. سپس با ایده‌آل‌ترین مسیر، به سمت مقصد حرکت میکند:



سپس به پیاده‌سازی الگوریتم Bug 2 میپردازیم. تابع‌های این قسمت نیز همانند سوال ۳ هستند با این تفاوت که تابع‌های زیر را اضافه دارند.

یک تابع به شکل زیر است که خط بین دو نقطه را بدست میاورد:

```
def get_line(p1, p2):
    slope = (p1[1] - p2[1])/(p1[0] - p2[0])
    bias = p1[1] - slope*p1[0]
    return slope, bias
```

تابع‌های زیر نیز از اسم آنها، مشخص هستند:

```
def distance_to_line(slope, bias, p):
    return abs(slope*p[0] - p[1] + bias) / sqrt(slope**2 + 1)

def is_on_line(p, acc=10**(-3)):
    return distance_to_line(STRAIGHT_LINE_SLOPE, STRAIGHT_LINE_BIAS, p) <= acc
```

در آخر نیز، تابع avoid_obstacle_velocities به شکل زیر، تغییر کرده است:

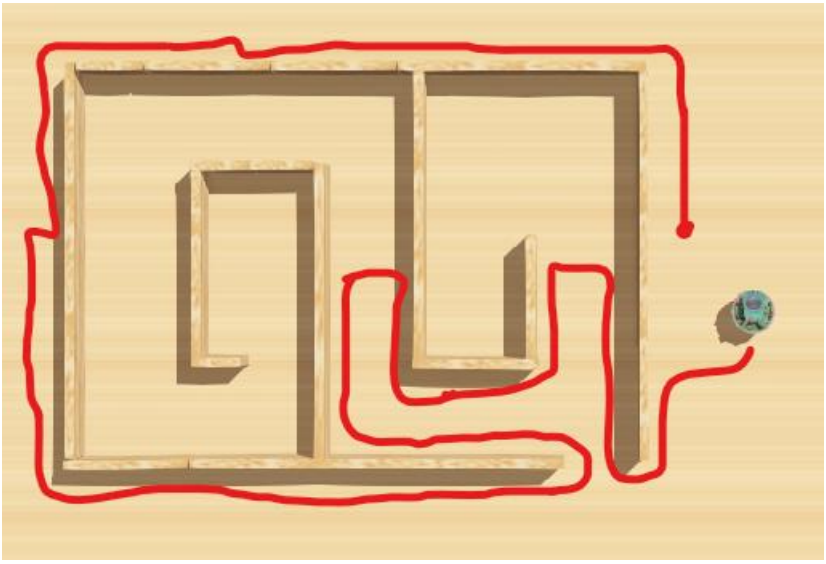
```
def avoid_obstacle_velocities():
    global turning_to_follow, follow_boundary, theta_at_start_of_turning
    global left_follow_dist, cnt, turning_in_the_end, go_in_line
    global K_P, DONT_CHANGE_K_P, TURN_POINTS
    obstacle = where_is_obstacle()

    cnt += 1

    if turning_to_follow:
        vl, vr = turn_left()
        theta = get_robot_heading(compass.getValues())
        if abs(theta - theta_at_start_of_turning) > 88:
            turning_to_follow = False
            follow_boundary = True

    elif follow_boundary:
        front, right, _, _, _ = get_distances()
```

```
    elif follow_boundary:
        front, right, _, _, _ = get_distances()

        if front <= DIST_THRESHOLD:
            turning_to_follow = True
            theta_at_start_of_turning = get_robot_heading(compass.getValues())
            follow_boundary = False

        if is_on_line(gps.getValues()[:2]):

            x, y = gps.getValues()[:2]
            been_here_before = False
            for TURN_POINT in TURN_POINTS:
                if dist((x, y), TURN_POINT) < 10**(-1):
                    # print('BEEN HERE BEFORE!')
                    been_here_before = True


            if not been_here_before and obstacle != 'LEFT':
                go_in_line = True
                # print('FOLLOWING STOPPED!', cnt)
                K_P = START_K_P
                DONT_CHANGE_K_P = True
                follow_boundary = False
                if dist((gx, gy), (x, y)) < dist((gx, gy), (TURN_POINTS[-1])):
                    TURN_POINTS.remove(TURN_POINTS[-1])
                TURN_POINTS.append((x, y))

        vl, vr = [FOLLOW_BOUNDARY_VELOCITY]*2
        vl -= K_F * (DIST_THRESHOLD - right)
        vr += K_F * (DIST_THRESHOLD - right)
```

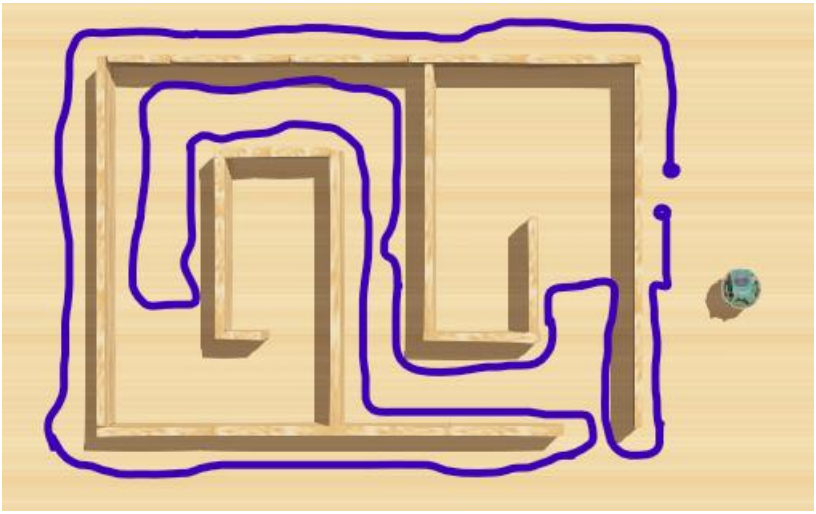
```
elif go_in_line:
    if obstacle == 'FRONT':
        go_in_line = False
    return get_velocities()



else:

    if obstacle is None:
        return get_velocities()

    elif obstacle == 'FRONT':
        vl, vr = handle_front_obstacle()

    elif obstacle == 'FRONT_LEFT':
        vl, vr = turn_right()

    elif obstacle == 'FRONT_RIGHT':
        vl, vr = turn_left()

    elif obstacle == 'LEFT':
        follow_boundary = True
        vl, vr = turn_right()

    elif obstacle == 'RIGHT':
        follow_boundary = True
        vl, vr = turn_left()

    else:
        return get_velocities()

return normalize(vl, vr)
```

الگوریتم Bug 2 بدین شکل عمل میکند که ابتدا با توجه به نقطه مبداء و مقصد تعیین شده، به سمت هدف حرکت میکند تا زمانی که به مانع برسد؛ وقتی که به مانع رسید، شروع به دنبال کردن مرزهای مانع میکند و همزمان خط بین نقطه فعلی با نقطه هدف را نیز چک میکند. زمانی که در راستای خط هدف قرار گرفت دیگر مرزهای مانع را دنبال نمیکند و میچرخد تا به سمت هدف حرکت کند و به همین ترتیب این رویه را انجام میدهد. در مواردی ممکن است چرخیدن ربات بهینه نباشد و در لوپ بیفتد؛ برای حل این مشکل ربات هنگام چرخش، جهت حرکت دیگر خود را درنظر میگیرد و در دفعه دوم که در موقعیت مشابه قرار گرفت، دیگر عمل چرخش را انجام نمیدهد و جهت حرکتی که در دفعه اول در نظر گرفته بود، ادامه میدهد. به همین ترتیب ادامه میدهد تا در آخر، به نقطه نهایی و هدف، میرسد.

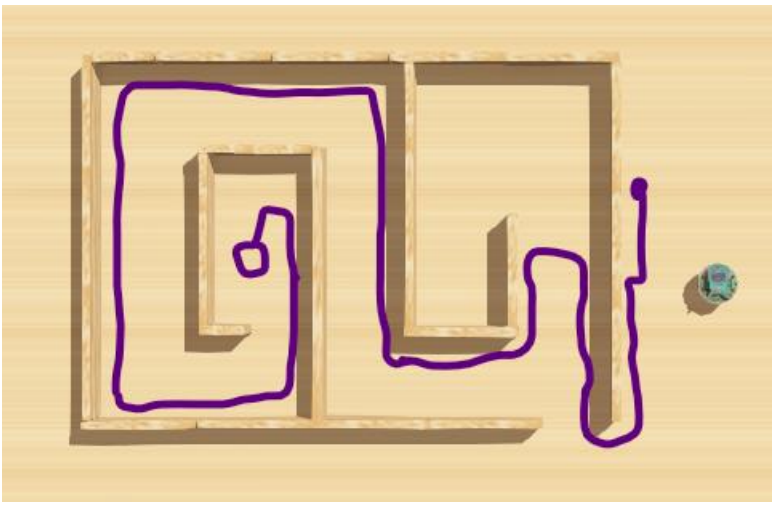
مسیر حرکت ربات به شکل زیر است:



چرخش‌های گفته شده در شکل بالا دیده میشود. ادامه‌ی مسیر ربات از نقطه‌ی پررنگ شده‌ی قرمز بالا به شکل زیر است:



همانطور که مشاهده میشود، براثر چرخش‌های انجام شده، ربات باز هم به همان نقطه‌ی شروع در تصویر بالا، رسیده است؛ اما این سری با توجه موردی که در توضیح الگوریتم گفته شد، مسیر حرکت دیگر را حفظ دارد تا در لوپ نیفتد:



بدین شکل، در این حرکت چرخش نابهینه ربات وجود ندارد و چرخش آخر ربات، منجر به رسیدن آن به مقصد شده است.

سوال ۲:

برای بدست آوردن نقشه محیط، از الگوریتم Bug 1 مطابق سوال یک، استفاده میکنیم و زمانی که ربات در حال پیمودن مانع‌ها است، نقطه‌ی هر لحظه از آن را ذخیره میکنیم و در نهایت، به کمک الگوریتم Split and Merge نقطه‌های مانع‌ها را به هم وصل کرده و نقشه‌ی محیط را بدست میاوریم.

کدهای آن شبیه به سوال یک است با این تفاوت که تابع‌های زیر را نیز دارد:

در تابع زیر، نقاط موانع را بدست میاوریم:

```
OBSTACLE_POINTS = []
def gather_data():
    front, right, left, _, _ = get_distances()
    x, y, _ = gps.getValues()
    theta = get_robot_heading(compass.getValues())
    theta *= PI/180

    if right < DIST_THRESHOLD*2:
        obstacle_point = (right*cos(theta-PI/2) + x, right*sin(theta-PI/2) + y)
        OBSTACLE_POINTS.append(obstacle_point)
```

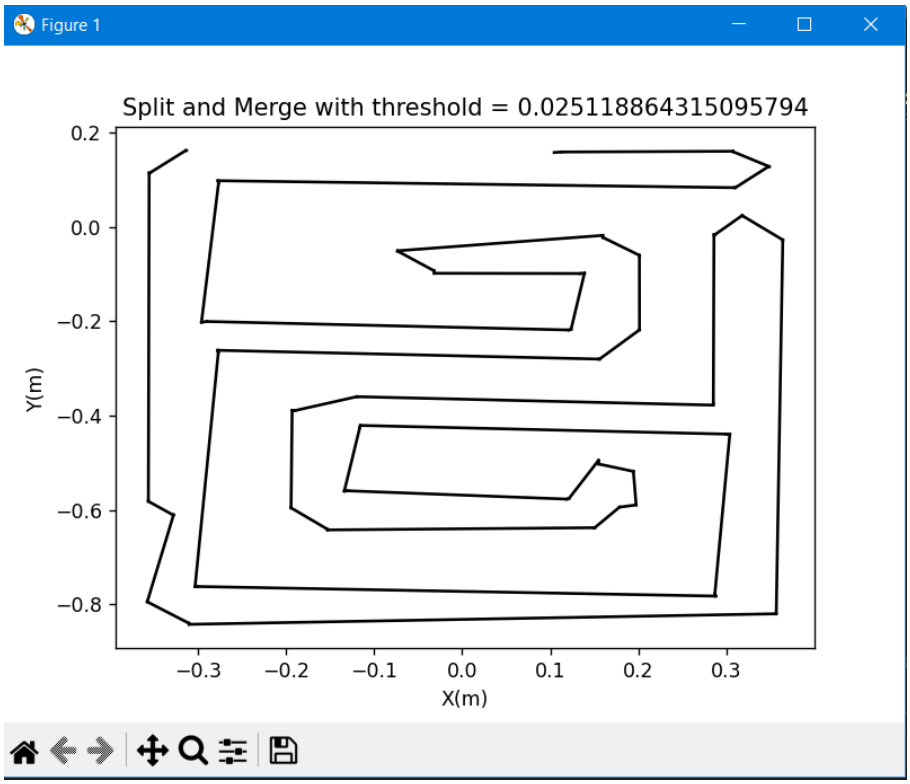
در تابع زیر نیز، عمل Split را انجام میدهیم:

```
LIST_OF_LINES = []
def split(points, thresh):

    slope, bias = get_line(points[0], points[-1])

    dists = [distance_to_line(slope, bias, point) for point in points]

    max_index = np.argmax(dists)

    if dists[max_index] > thresh:
        split(points[:max_index+1], thresh)
        split(points[max_index:], thresh)

    else:
        LIST_OF_LINES.append((points[0], points[1]))
```

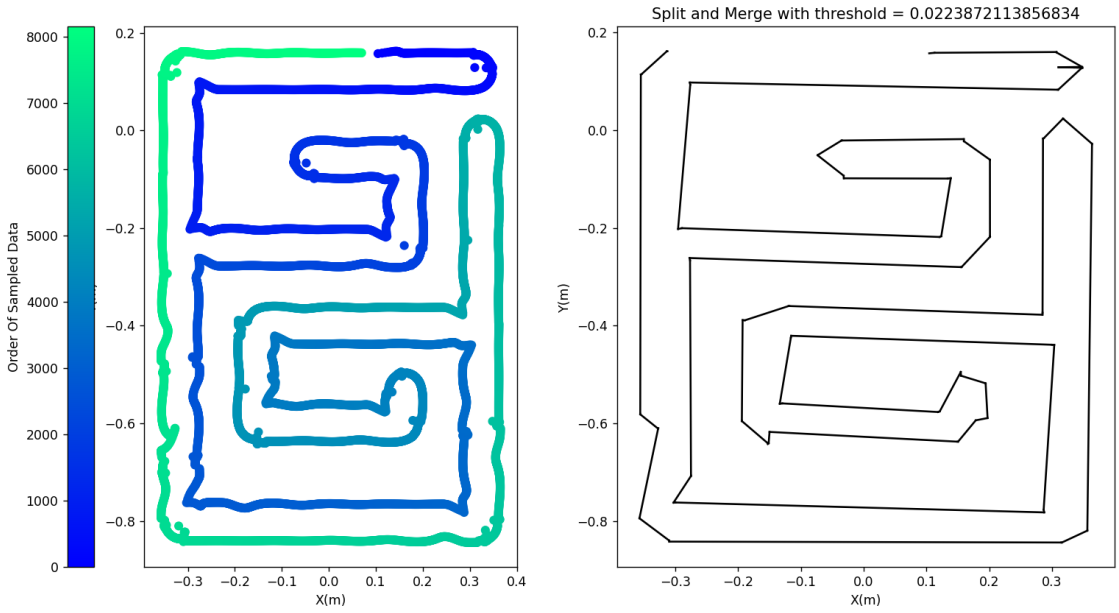
در آخر نیز، عمل Merge انجام میشود و خطوط موانع بدست میایند:

```
THRESHOLD = 10**(-1.6)
def split_merge():
    split(OBSTACLE_POINTS, thresh = THRESHOLD)

    plt.title(f'Split and Merge with threshold = {THRESHOLD}')
    plt.xlabel('X(m)')
    plt.ylabel('Y(m)')


    # merge:
    for i in range(len(LIST_OF_LINES)):

        p0 = LIST_OF_LINES[i][0]
        p1 = LIST_OF_LINES[i][1]
        try:
            p2 = LIST_OF_LINES[(i+1)][0]
        except:
            p2 = p1
        plt.plot([p0[0], p1[0]], [p0[1], p1[1]], c='black')
        plt.plot([p1[0], p2[0]], [p1[1], p2[1]], c='black')

    plt.show()
```

شکل نمودار بالا نیز، به صورت زیر است:



نمایش دیگری نیز برای نقشه‌ی محیط تهیه کرده‌ایم که به صورت زیر است:



در نهایت، نقشه‌ی محیط بدست آمد.

سوال ۳:

برای پیاده‌سازی الگوریتم تعقیب دیوار، به شکل زیر عمل کنیم:

ابتدا برای بدست آوردن سر ربات، از تابع زیر استفاده میکنیم:

```
def get_robot_heading(compass_value):
    rad = atan2(compass_value[1], compass_value[0])
    bearing = (rad - PI/2) / PI * 180.0
    if bearing < 0.0:
        bearing = bearing + 360.0

    heading = 360 - bearing
    if heading > 360.0:
        heading -= 360.0
    return heading
```

سپس به کمک تابع زیر، اطمینان حاصل میکنیم که زاویه بین رنج صفر تا 2π قرار داشته باشد:

```
def go_in_upper_circle(angle: float) -> float:
    if angle > PI:
        angle -= 2 * PI
    elif angle < -PI:
        angle += 2 * PI
    return angle
```

در ادامه مقدار فاصله نقطه فعلی ربات تا مقصد مدنظر را، به کمک تابع زیر در مختصات قطبی، بدست میاوریم:

```
def get_polar_error() -> list[float, float, float]:
    x, y, _ =   gps.getValues()
    teta = get_robot_heading(compass.getValues())*PI/180

    p = dist((gx, gy), (x, y))
    a = -teta + atan2(gy - y, gx - x)
    b = -teta-a

    b, a = go_in_upper_circle(b), go_in_upper_circle(a)
    return p, a, b
```

سپس به کمک تابع زیر، سرعت چرخ چپ و راست ربات را بین رنج 2π و -2π نرمالایز میکنیم زیرا ربات e-puck بین این بازه سرعتی کار میکند:

```
def normalize(vl, vr):
    if not (-2*PI<=vl<=2*PI) or not (-2*PI<=vr<=2*PI):
        coeff = (2*PI-0.1)/ max(abs(vl), abs(vr))
        vl *= coeff
        vr *= coeff
        print(vl, vr)
    return vl, vr
```

در آخر، به کمک تابع زیر، سرعت چرخ چپ و راست ربات را تعیین میکنیم که در آن، ضرایب کنترلر (K_P، K_A و K_B) نیز مشخص هستند:

```
def get_velocities() -> list[float, float]:
    p, a, b = get_polar_error()
    global K_P
    v: float = K_P*p
    w: float = K_A*a + K_B*b

    if K_P < 6-INCREMENT_K_P:
        K_P += INCREMENT_K_P

    v_left = (v - L*w/2)
    v_right = (v + L*w/2)

    return normalize(v_left, v_right)
```

اکنون سنسورهای مورد نیاز برای ربات را به صورت زیر مشخص میکنیم:

```
gps = robot.getDevice("gps")
compass = robot.getDevice("compass")
f = robot.getDevice('front distance sensor')
fr = robot.getDevice('front right distance sensor')
fl = robot.getDevice('front left distance sensor')
r = robot.getDevice('right distance sensor')
l = robot.getDevice('left distance sensor')
```

به کمک این سنسورها، در تابع زیر، فاصله‌های هر بخش را تعیین میکنیم:

```
def get_distances():
    front = f.getValue()
    left = l.getValue()
    right = r.getValue()
    front_right = fr.getValue()
    front_left = fl.getValue()

    return front, right, left, front_right, front_left
```

در ادامه با توجه به فاصله‌های هر سمت، وجود مانع را تشخیص میدهیم:

```
def where_is_obstacle():
    strings = ['FRONT', 'RIGHT', 'LEFT', 'FRONT_RIGHT', 'FRONT_LEFT']
    dists = get_distances()
    for i in range(len(dists)):
            if dists[i] <= DIST_THRESHOLD:
                return strings[i]
    return None
```

چرخش چرخ‌ها نیز، به شکل زیر است:

```
def turn_right():
    return TURN_VELOCITY, -TURN_VELOCITY

def turn_left():
    return -TURN_VELOCITY, TURN_VELOCITY
```

مانع جلوی ربات، به کمک تابع زیر بدست میاید:

```
def handle_front_obstacle():
    global turning_to_follow, follow_boundary, theta_at_start_of_turning
    global left_follow_dist
    vl, vr = turn_left()
    turning_to_follow = True
    theta_at_start_of_turning = get_robot_heading(compass.getValues())
    K_P = START_K_P
    return vl, vr
```

در آخر، به کمک تابع زیر navigation ربات را تعیین میکنیم:

```
def avoid_obstacle_velocities():
    global turning_to_follow, follow_boundary, theta_at_start_of_turning
    global left_follow_dist, cnt, turning_in_the_end, chase_goal
    obstacle = where_is_obstacle()
```

سه استیت داریم:

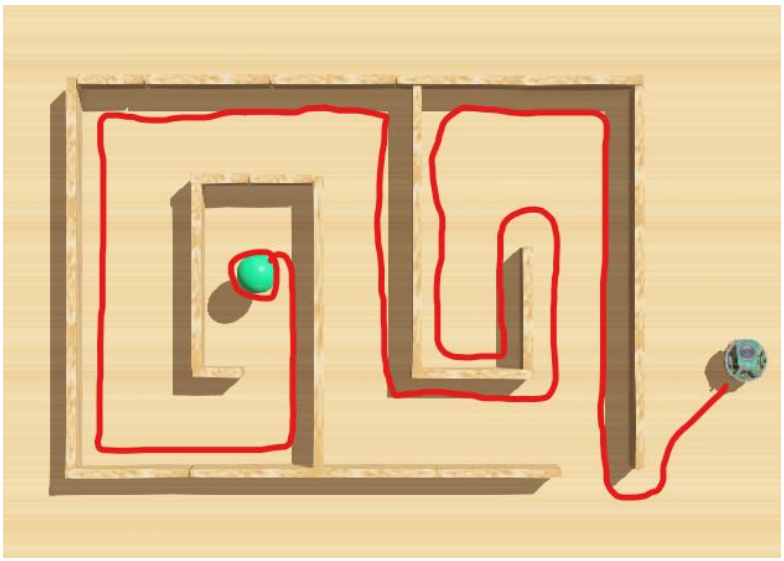
- ربات به مسیر مستقیم ادامه دهد.
- ربات تغییر جهت دهد.
- ربات مرزهای مانع‌ها را دنبال میکند.

```
    cnt += 1
    if turning_to_follow:
        print(f'TURNING {cnt}')
        vl, vr = turn_left()
        theta = get_robot_heading(compass.getValues())
        if abs(theta - theta_at_start_of_turning) > 88:
            turning_to_follow = False
            print(f'END OF TURN {cnt}')
            follow_boundary = True

    elif follow_boundary:
        print(f'FOLLOWING_BOUNDARY {cnt}')
        front, right, _, _, _ = get_distances()

        if front <= DIST_THRESHOLD:
            turning_to_follow = True
            theta_at_start_of_turning = get_robot_heading(compass.getValues())
            follow_boundary = False

        vl, vr = [FOLLOW_BOUNDARY_VELOCITY]*2
        vl -= K_F * (DIST_THRESHOLD - right)
        vr += K_F * (DIST_THRESHOLD - right)
```

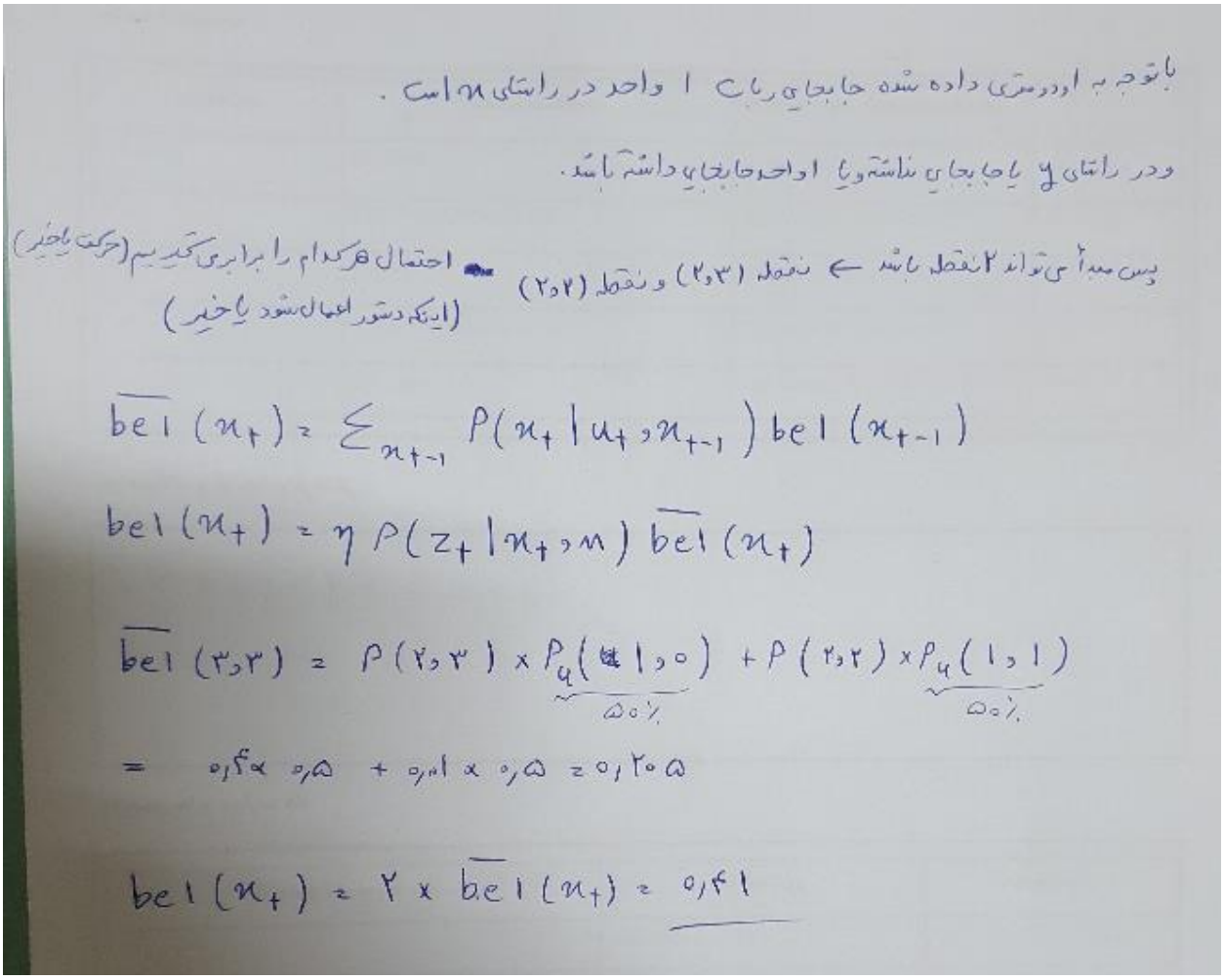
```
        if dist(gps.getValues()[:2], (gx, gy)) < 10**(-1):
            print('CHASING GOAL')
            follow_boundary = False
            chase_goal = True

    elif chase_goal:
        return get_velocities()
```

```
    else:

        if obstacle is None:
            print(f'No OBSTACLES {cnt}')
            return get_velocities()

        elif obstacle == 'FRONT':
            print('FRONT')
            vl, vr = handle_front_obstacle()

        elif obstacle == 'FRONT_LEFT':
            print('FRONT_LEFT')
            vl, vr = turn_right()

        elif obstacle == 'FRONT_RIGHT':
            print('FRONT_RIGHT')
            vl, vr = turn_left()

        elif obstacle == 'LEFT':
            print('LEFT')
            follow_boundary = True
            vl, vr = turn_right()

        elif obstacle == 'RIGHT':
            follow_boundary = True
            vl, vr = turn_left()

        else:
            print(obstacle)
            return get_velocities()

    return normalize(vl, vr)
```

مسیر حرکت ربات به صورت زیر است:



سوال ۴:


با توجه به اودومتری داده شده جابجایی ربات ۱ واحد در راستای x است.
ودر راستای y جابجایی نداشته و یا اواحد جابجایی داشته باشد.
پس میدانی تواند ۲ نقطه باشد ← نقطه (۲،۳) و نقطه (۲،۲)
احتمال هرکدام را برابری گذیم (حرکت یا خیر)
(اینکه دستور اعمال شود یا خیر)
$\overline{bel}(x_t) = \sum_{x_{t-1}} P(x_t | u_t, x_{t-1}) bel(x_{t-1})$
$bel(x_t) = \eta P(z_t | x_t, m) \overline{bel}(x_t)$
$\overline{bel}(۳،۳) = P(۲،۳) \times P_u(\text{یک} | ۰،۰) + P(۲،۲) \times P_u(۱،۱)$
۵۰٪
۵۰٪
$= ۰،۶ \times ۰،۵ + ۰،۱ \times ۰،۵ = ۰،۳۵$
$bel(x_t) = ۲ \times \overline{bel}(x_t) = ۰،۷۱$