```
from __future__ import print_function
import rospy

import random
import numpy as np

import robot_global
import human
import motor
import laser
import time_step
import math
import tf
import sys
from time import gmtime, strftime
import copy
import time



from ShortestPathFindAlgorithm import spfa
from ShortestPathFindAlgorithm import unit

from std_srvs.srv import Empty
from sensor_msgs.msg import LaserScan
from std_msgs.msg import String

from std_msgs.msg import String
from geometry_msgs.msg import Twist
from std_msgs.msg import Bool
from geometry_msgs.msg import Quaternion
from geometry_msgs.msg import Vector3

#ACTION_LOG_FILE = open("my_net18/actionLog_" + strftime("%Y-%m-%d-%H-%M-%S", gmtime()) + ".txt", 'a+')
actionoutPath = "my_net22/actionLog_" + strftime("%Y-%m-%d-%H-%M-%S", gmtime()) + ".txt"
PI=3.1415926
RSTATEDIS = 10
RSTATEANGLE = 10
RSTATEFACINGANGLE = 10
RREWARDDIS = 20*5
RREWARDANGLE = 20
RREWARDFACINGANGLE = 20*4
FACINGDIS_RANGE = 1
```

真实数据到网络输入状态时的比例

→ 真实数据→给直接reward的比例

一米之内再考虑Facing

```
MAX_LASER_RANGE = 3
MIN_LASER_RANGE = 0
```

[0,3]之内雷达才接收

```
MINNUM = 0.1
FAILREWARD = -300
WINREWARD = 500
```

奖励

```
ARRIVEDIS = 0.8
ARRIVEANGLE = 0.5
```

到达目标要求

```
POSITIVEDIS = 0.8
DISFROMSTARTTOEND = 4
MINDISFROMSTARTTOEND = 0.5
```

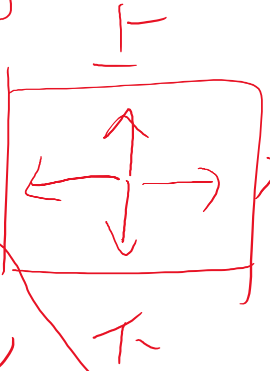
初始化位置用(简单)

```
my_case_robotx = [3, 1, 3, 2, 2, 1]
my_case_roboty = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
my_case_robotz = [2, 2, 2.5, 2.5, 2.5, 7]

my_case_robot_rotat_x = [0, 0, 0, 0, 0, 0]
my_case_robot_rotat_y = [1, 1, 1, 1, 1, 1]
my_case_robot_rotat_z = [0, 0, 0, 0, 0, 0]
my_case_robot_rotat_w = [PI, 0, -PI, -PI / 4, -PI * 3 / 4, -PI / 4]

my_case_humanx = [3, 3, 3, 2, 2, 3]
my_case_humany = [1.27, 1.27, 1.27, 1.27, 1.27, 1.27]
my_case_humanz = [8, 8, 6, 7, 7, 3.5]

opposite_x = [0, 0, 0, 2, 2, 0]
opposite_z = [-1.5, -1.5, 1.5, 0, 0, -1.5]
```

没用

```
tmp_out_pos = 0
global_rn = 0
```

用来确定左右(相对比较麻烦)



```
def get_human_position_res_callback(position_value):
        human.position = position_value
        human.get_position_res_flag = True


def pub_get_human_position_req():
        value = Bool()
        value.data = True
        human.PubGetPositionReq.publish(value)
        human.get_position_res_flag = False
```

但用的是为了知道机器人到目的地和人的距离，可重写该调用的的判定

这都是的处理可忽略。

```
def get_human_rotation_res_callback(rotation_value):
        human.rotation = rotation_value
        human.get_rotation_res_flag = True


def pub_get_human_rotation_req():
        value = Bool()
        value.data = True
        human.PubGetRotationReq.publish(value)
        human.get_rotation_res_flag = False


def get_position_res_callback(position_value):
        robot_global.position = position_value
        robot_global.get_position_res_flag = True


def pub_get_position_req():
        value = Bool()
        value.data = True
        robot_global.PubGetPositionReq.publish(value)
        robot_global.get_position_res_flag = False


def set_position_res_callback(value):
        robot_global.set_position_res_flag = value.data

## new add for human
def set_human_pose_res_callback(value):
        human.set_human_pose_res_flag = value.data

def pub_set_human_pose_req(position_value):
        human.PubSetPositionReq.publish(position_value)
        human.set_human_pose_res_flag = False

def pub_set_position_req(position_value):
        robot_global.PubSetPositionReq.publish(position_value)
        robot_global.set_position_res_flag = False

def set_rotation_res_callback(value):
        robot_global.set_rotation_res_flag = value.data

def pub_set_rotation_req(rotation_value):
        robot_global.PubSetRotationReq.publish(rotation_value)
        robot_global.set_rotation_res_flag = False

def get_rotation_res_callback(rotation_value):
        robot_global.rotation = rotation_value
        robot_global.get_rotation_res_flag = True


def pub_get_rotation_req():
        value = Bool()
        value.data = True
        robot_global.PubGetRotationReq.publish(value)
        robot_global.get_rotation_res_flag = False


def reset_node_physics_res_callback(value):
        robot_global.reset_node_physics_res_flag = value.data


def pub_reset_node_physics_req():
        value = Bool()
        value.data = True
        robot_global.PubResetNodePhsicsReq.publish(value)
        robot_global.reset_node_physics_res_flag = False



def human_connect():
        rospy.Subscriber("/human_name", String)
        # connect
        model_name = None
        while model_name is None:
            try:
                model_name = rospy.wait_for_message('/human_name', String, timeout=5)
            except:
                pass
        print("human %s connect success" % model_name.data)
        return model_name.data

def robot_connect():
        rospy.Subscriber("/model_name", String)
        # connect
        model_name = None
        while model_name is None:
            try:
                model_name = rospy.wait_for_message('/model_name', String, timeout=5)
            except:
                pass
        print("robot %s connect success" % model_name.data)
        return model_name.data
```

```
#change webots laser and dist to our focus data
def discretize_observation(laser_data, laser_dim, collision_threshold):
    sum = 0.0
    new_laser_data = []

    for i, item in enumerate(laser_data.ranges):
        sum += laser_data.ranges[i]
        if ((i + 1) % 3 == 0):
            new_laser_data.append(sum / 3.0)
            sum = 0

    collision = False

    mod = len(new_laser_data) / laser_dim
    discretized_ranges = []
    ###just for habit..
    HJKINF = 1000000
    tmp_cnt = 0
    min_val = HJKINF
    for i, item in enumerate(new_laser_data):
        ####add
        if (i % mod == mod - 1 and tmp_cnt + 1 <= laser_dim):
            min_val = min(min_val, new_laser_data[i])
            tmp_cnt = tmp_cnt + 1
            if min_val > MAX_LASER_RANGE:
                discretized_ranges.append(MAX_LASER_RANGE)
            elif min_val < MIN_LASER_RANGE:
                discretized_ranges.append(MIN_LASER_RANGE)
            else:
                discretized_ranges.append(min_val)
        ####init
        if (i % mod == 0):
            min_val = HJKINF
            min_val = new_laser_data[i]
        #####update
```

⇒ 取有用的雷达信息

```
        min_val = min(min_val, new_laser_data[i])

        if (new_laser_data[i] < collision_threshold):
            collision = True
    print("hjk--- check dim of laser: ", len(discretized_ranges))
    #for x in enumerate(discretized_ranges):
     #    print("***check**: ", x)
    return discretized_ranges, collision


#get angle from point 1 to point 2
#angle is rotate y-axis(+) to vector<1=>2>, reverse clock is +(0, PI), clock is -(0, PI)
def getangle(x1, y1, x2, y2):
        if y1 > y2 + MINNUM:
                rtang = math.atan((x1-x2)/(y1-y2))
        elif math.fabs(y1-y2) < MINNUM:
                if x1 > x2+MINNUM:
                        rtang = PI/2
                elif x1 < x2-MINNUM:
                        rtang = -PI/2
                else:
                        rtang = 0
        else:
                if x1 > x2:
                        rtang = PI+math.atan((x1-x2)/(y1-y2))
                else:
                        rtang = -PI+math.atan((x1-x2)/(y1-y2))

        return rtang
```

→ 计算角度，需要用纸画一画，比较绕（这是用的之前的代码，当时我也用笔画了很久，现在记不清楚了……）

```
#get feature between point 1 and point 2: angle from point 1 to interleave angle, angle from
interleave angle to point 2
def getanglefea(x1, y1, w1, x2, y2, w2):
        rtang = getangle(x1, y1, x2, y2)

        ang1 = rtang - w1
        if ang1 > PI:
                ang1 = ang1-2*PI
        elif ang1 <= -PI:
                ang1 = ang1 + 2*PI



        ang2 = w2 - w1
        if ang2 > PI:
                ang2 = ang2 - 2*PI
        elif ang2 <= -PI:
                ang2 = ang2 + 2*PI

        return rtang, ang1, ang2


#now for hjk, just calcu dist.
def getDisXZ(robotpos_x, robotpos_z, robotrot_w, rightpos_x, rightpos_z, rightrot_w):
    distx = robotpos_x - rightpos_x
    distz = robotpos_z - rightpos_z
    dist1 = math.sqrt(distx * distx + distz * distz)
    return dist1



def getopposite(position, rotation, my_case = -1):
        rpos = Vector3()
        rpos.x = position.x
        rpos.y = position.y

        rpos.z = position.z #- 0.5

        global tmp_out_pos
        tmp_out_pos = 0
        global global_rn
        if my_case == -1:
            print("hjktest---- getopposit this")
            #rn = random.random()
            if global_rn < 0.25:
                rpos.x = position.x - POSITIVEDIS
                tmp_out_pos = 1
            elif global_rn < 0.5:
                rpos.x = position.x + POSITIVEDIS
                tmp_out_pos = 3
            elif global_rn < 0.75:
                rpos.z = position.z + POSITIVEDIS
                tmp_out_pos = 2
            else:
                rpos.z = position.z - POSITIVEDIS
                tmp_out_pos = 4
        else:
            rpos.x = rpos.x + opposite_x[my_case]
            rpos.z = rpos.z + opposite_z[my_case]
            if opposite_x[my_case] > 0.01:
```

⇒ 得到目标位置

这指的是随机初始位置的情况下

固定位置

（建议：初始化位置和抵到目的的代码重新写，因为当时既要打表初始位置，又要随机初始位置比较乱）

```
                tmp_out_pos = 3
            if opposite_x[my_case] < -0.01:
                tmp_out_pos = 1
            if opposite_z[my_case] > 0.01:
                tmp_out_pos = 2
            if opposite_z[my_case] < -0.01:
                tmp_out_pos = 4

        print("GGGGGGGGGg************************* goal is : ", tmp_out_pos)
        #rpos.z = position.z + 3    # - 0.5

        rrot = Quaternion()
        rrot.x = rotation.x
        rrot.y = rotation.y
        rrot.z = rotation.z
        rrot.w = PI
        if tmp_out_pos == 1:
            rrot.w = -PI/ 2
        elif tmp_out_pos == 2:
            rrot.w = 0
        elif tmp_out_pos == 3:
            rrot.w = PI / 2
        elif tmp_out_pos == 4:
            rrot.w = PI


       # if rrot.w > PI:
        #           rrot.w = rrot.w - PI*2

        return rpos, rrot


def changeStateFromEnvToNetwork(laser_state, dist1, rot1, rot2, action):
    print("hjk--- rot1 is : ", rot1)
    print("hjk--- dist1 is : ", dist1)
    # no need to add do something to laser_state. Becuase laser_state has its
limit !!!!!!!!!!!!!!!!!!!
    limitLaser_state = laser_state
    state = limitLaser_state + [dist1 * RSTATEDIS, rot1 * RSTATEANGLE, rot2 *
RSTATEFACINGANGLE, action]
    return state
```

|没用

```
class WebotsLidarNnEnv(): #why not object?? hjk
    def __init__(self, laser_dim, collision_threshold):
        self.my_case = -1
        global global_rn
        global_rn = random.random()
        self.laser_dim = laser_dim
        self.collision_threshold = collision_threshold

        rospy.init_node('webots_env', anonymous=True)

        robot_global.robot_name = robot_connect()
        human.human_name = human_connect()

        robot_global.PubSetPositionReq   =   rospy.Publisher('/simulation_set_position_req',
Vector3, queue_size=1)
        robot_global.SubSetPositionRes   =   rospy.Subscriber('/simulation_set_position_res',
Bool,

set_position_res_callback)

        ###new add for human_pose
        human.PubSetPositionReq    =    rospy.Publisher('/simulation_set_human_pose_req',
Vector3, queue_size=1)
        human.SubSetPositionRes    =    rospy.Subscriber('/simulation_set_human_pose_res',
Bool,

set_human_pose_res_callback)
       #    human.PubSetRotationReq    =    rospy.Publisher('/simulation_set_rotation_req',
Quaternion, queue_size=1)
        #human.SubSetRotationRes = rospy.Subscriber('/simulation_set_rotation_res', Bool,
         #
set_rotation_res_callback)

        robot_global.PubSetRotationReq   =   rospy.Publisher('/simulation_set_rotation_req',
Quaternion, queue_size=1)
        robot_global.SubSetRotationRes   =   rospy.Subscriber('/simulation_set_rotation_res',
Bool,
                                                              set_rotation_res_callback)

        robot_global.PubResetNodePhsicsReq                                                    =
rospy.Publisher('/simulation_reset_node_physics_req', Bool, queue_size=1)
        robot_global.SubResetNodePhsicsRes                                                    =
rospy.Subscriber('/simulation_reset_node_physics_res', Bool,
```

初始化信息，没用

```
reset_node_physics_res_callback)

        robot_global.PubGetPositionReq = rospy.Publisher('/simulation_get_position_req',
Bool, queue_size=1)
        robot_global.SubGetPositionRes = rospy.Subscriber('/simulation_get_position_res',
Vector3,

get_position_res_callback)

        robot_global.PubGetRotationReq = rospy.Publisher('/simulation_get_rotation_req',
Bool, queue_size=1)
        robot_global.SubGetRotationRes = rospy.Subscriber('/simulation_get_rotation_res',
Quaternion,

get_rotation_res_callback)

        human.PubGetPositionReq = rospy.Publisher('/simulation_get_human_position_req',
Bool, queue_size=1)
        human.SubGetPositionRes                                                    =
rospy.Subscriber('/simulation_get_human_position_res', Vector3,

get_human_position_res_callback)

        human.PubGetRotationReq                                                    =
rospy.Publisher('/simulation_get_human_rotation_req', Bool, queue_size=1)
        human.SubGetRotationRes                                                    =
rospy.Subscriber('/simulation_get_human_rotation_res', Quaternion,

get_human_rotation_res_callback)

        for i in range(0, 5):
            time_step.time_step_call() #zuoyong? hjk

        motor.init()
        motor.set_velocity(0, 0, 0, 0)

        time_step.time_step_call()
        time_step.time_step_call()
        time_step.time_step_call()

        laser.init()
        #laser.get_laser_scan_data()

        self.reward_range = (-np.inf, np.inf)
```

} 相关也没太在意，
是和VOS交互的

```
          self.action_history1 = 0
          self.action_history2 = 0
          self.action_history3 = 0


          for i in range(0, 5):
               time_step.time_step_call()

          #get robot pose
          pub_get_position_req()
          while robot_global.get_position_res_flag is False:
                    time_step.time_step_call()
          pub_get_rotation_req()
          while robot_global.get_rotation_res_flag is False:
                    time_step.time_step_call()

          #get human pose
          pub_get_human_position_req()

          while human.get_position_res_flag is False:
                    time_step.time_step_call()
          pub_get_human_rotation_req()
          while human.get_rotation_res_flag is False:
                    time_step.time_step_call()

          rpos, rrot = getopposite(human.position, human.rotation, self.my_case)
          dist1        =        getDisXZ(robot_global.position.x,        robot_global.position.z,
robot_global.rotation.w, rpos.x, rpos.z, rrot.w)
          rtang,  rot1,  rot2  =  getanglefea(robot_global.position.x,  robot_global.position.z,
robot_global.rotation.w,
                                                  rpos.x, rpos.z, rrot.w)
          self.distp = dist1 #past distance
          self.rot1p =rot1
          self.rot2p = rot2
          self.init_dist_pos = 0
          self.wintimes_dist_key = 10
          self.wintimes_all = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
          self.wintimes_win = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
          self.wintimes = 0
          self.collisiontimes = 0
          self.N = int(6.6 * unit + 10)
          self.M = int(9.9 * unit + 10)
          self.mp = self.init_map_hjktest()
```

后面阶段用，如最优路径相关。

```
        self.lenp = 0
        self.old_robot_postion = copy.deepcopy(robot_global.position)
        self.old_robot_rotation = copy.deepcopy(robot_global.rotation)



    def init_map_hjktest(self):
        mp = np.zeros((self.N, self.M))
        for i in range(int(4.5 * unit), int(6.60 * unit + 1)):
            for j in range(int(0 * unit), int(1.90 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(6.00 * unit), int(6.60 * unit + 1)):
            for j in range(int(2.80 * unit), int(5.00 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(2.30 * unit), int(3.70 * unit + 1)):
            for j in range(int(3.15 * unit), int(3.45 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(3.70 * unit), int(4.10 * unit + 1)):
            for j in range(int(9.10 * unit), int(9.90 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(1.60 * unit), int(3.40 * unit + 1)):
            for j in range(int(8.50 * unit), int(9.90 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(0 * unit), int(1.00 * unit + 1)):
            for j in range(int(0 * unit), int(3.30 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(0 * unit), int(1.00 * unit + 1)):
            for j in range(int(4.40 * unit), int(5.80 * unit + 1)):
                mp[i][j] = 1
        for i in range(int(0 * unit), int(1.40 * unit + 1)):
            for j in range(int(6.10 * unit), int(7.90 * unit + 1)):
                mp[i][j] = 1
        return mp

    def distance(self, p1, p2):
        dist = math.sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y)+(p1.z-p2.z)*(p1.z-
p2.z))
        return dist
```

没用，最短路
什么地图

```
def step(self, action):

    if action == 0:    # FORWARD
        if self.action_history1 != 0:
            motor.set_velocity(0, 0, 0, 0)
            pub_reset_node_physics_req()
            while robot_global.reset_node_physics_res_flag is False:
                time_step.time_step_call()
        motor.set_velocity(5.0, 5.0, 5.0, 5.0)

    elif action == 1:    # LEFT forward
        if self.action_history1 != 3:
            motor.set_velocity(0, 0, 0, 0)
            pub_reset_node_physics_req()
            while robot_global.reset_node_physics_res_flag is False:
                time_step.time_step_call()
        motor.set_velocity(4.0, 4.0, 7.0, 7.0)

    elif action == 2:    # RIGHT forward
        if self.action_history1 != 4:
            motor.set_velocity(0, 0, 0, 0)
            pub_reset_node_physics_req()
            while robot_global.reset_node_physics_res_flag is False:
                time_step.time_step_call()
        motor.set_velocity(7.0, 7.0, 4.0, 4.0)

    elif action == 3:    # TURN LEFT
        if self.action_history1 != 1:
            motor.set_velocity(0, 0, 0, 0)
            pub_reset_node_physics_req()
            while robot_global.reset_node_physics_res_flag is False:
                time_step.time_step_call()
        motor.set_velocity(-3.0, -3.0, 3.0, 3.0)

    elif action == 4:    # TURN RIGHT
        if self.action_history1 != 2:
            motor.set_velocity(0, 0, 0, 0)
            pub_reset_node_physics_req()
            while robot_global.reset_node_physics_res_flag is False:
                time_step.time_step_call()
        motor.set_velocity(3.0, 3.0, -3.0, -3.0)
        self.action_history3 = self.action_history2
        self.action_history2 = self.action_history1
        self.action_history1 = action

        for i in range(0, 4):
            time_step.time_step_call()



        laser_data, done = laser.get_laser_scan_data()
        while done is False:
            laser_data, done = laser.get_laser_scan_data()
            time_step.time_step_call()

        laser_state, is_collision = discretize_observation(laser_data, self.laser_dim,
self.collision_threshold)
        # get robot pose
        pub_get_position_req()
        while robot_global.get_position_res_flag is False:
            time_step.time_step_call()
        pub_get_rotation_req()
        while robot_global.get_rotation_res_flag is False:
            time_step.time_step_call()

        # get human pose
        pub_get_human_position_req()
        while human.get_position_res_flag is False:
            time_step.time_step_call()
        pub_get_human_rotation_req()
        while human.get_rotation_res_flag is False:
            time_step.time_step_call()

        # get right opposite pose to human
        rpos, rrot = getopposite(human.position, human.rotation, self.my_case)

        rtang, rot1, rot2 = getanglefea(robot_global.position.x, robot_global.position.z,
robot_global.rotation.w, rpos.x, rpos.z, rrot.w)

        dist1 = getDisXZ(robot_global.position.x, robot_global.position.z,
robot_global.rotation.w, rpos.x, rpos.z,
                         rrot.w)
        # state = laser_state + [dist1, rot1, rot2] + action_history

        #print ("from robot to person: ", robot_global.position.x, robot_global.position.z,
robot_global.rotation.w,
         #           human.position.x, human.position.z, human.rotation.w, dist1, rot1)
        # print ("State : " + str(state) + " Action : " + str(action))



        state = changeStateFromEnvToNetwork(laser_state, dist1, rot1, rot2, action)
        '''
        distance = getDisXZ(robot_global.position.x, robot_global.position.z,
robot_global.rotation.w, self.old_robot.position.x, self.old_robot.position.z,
self.old_robot.rotation.w)
        #distance = getDisXZ(robot_global.position.x, robot_global.position.z,
self.old_robot.position.x, self.old_robot.position.z, self.old_robot.rotation.w)
        ACTION_LOG_FILE = open("my_net18/actionLog_" +
strftime("%Y-%m-%d-%H-%M-%S", gmtime()) + ".txt", 'a+')
        #print("???", file = ACTION_LOG_FILE)

        print("postion change :    ", distance, file=ACTION_LOG_FILE)
        doubletmp = 1.0 * (robot_global.rotation.w - self.old_robot.rotation.w)
        if doubletmp >= 2.0 * PI - 0.000001:
            doubletmp = 0
        if doubletmp <= -2.0 * PI + 0.000001:
            doubletmp = 0
        print("rotation value change :    ", doubletmp, file=ACTION_LOG_FILE)
        sys.stdout.flush()

        self.old_robot_x = copy.deepcopy(robot_global)
        '''

        done = is_collision
        if done is True :
            reward = FAILREWARD
            self.collisiontimes += 1
            print("--------------------------------------------------------------
-----------------")
            print("--------------------------------------------------------------
-----------------")
            print("--------------------------------------------------------------
-----------------")
            print("--------------------------------------------------------------
-----------------")
```

```
                        print("------------------------------------------------------------
-----------------")
                        print("------------------------------------------------------------
-----------------")
                        print("---------------------------------")
                        print("---------------------------------")
                        print ("-----NO! You collide it!------------")
                        print("---------------------------------")
                        print("---------------------------------")
                        print("---------------------------------")
                        print("------------------------------------------------------------
-----------------")
                        print("------------------------------------------------------------
-----------------")
                        print("------------------------------------------------------------
-----------------")
                        print("------------------------------------------------------------
-----------------")
                        print("------------------------------------------------------------
-----------------")
                else :
                            reward = 0
                            tmp_par = dist1
                            if tmp_par > FACINGDIS_RANGE:
                                    tmp_par = FACINGDIS_RANGE
                            print("hjk--- robot:    ", robot_global.rotation.w, rrot.w)
                            print("hjk--- delta dis last - now : ", self.distp - dist1)
                            print("hjk---   delta   angle   last   -   now   :   ",   math.fabs(self.rot1p),
math.fabs(rot1), math.fabs(self.rot1p) - math.fabs(rot1))
                            print("hjk--- 2222222   delta angle last - now : ", math.fabs(self.rot2p),
math.fabs(rot2), math.fabs(self.rot2p) - math.fabs(rot2))
                            if dist1 < 2.00:
                                    reward   =   tmp_par   *   ((self.distp   -   dist1)   *   RREWARDDIS   +
(math.fabs(self.rot1p)  -  math.fabs(rot1))  *  RREWARDANGLE)  +  (FACINGDIS_RANGE  -
tmp_par)  *  RREWARDFACINGANGLE  *  (math.fabs(self.rot2p)  -  math.fabs(rot2))  #  0  +
(self.difap - difa1)*10
                                    print ("hjk--reward:    ", tmp_par * ((self.distp - dist1) * RREWARDDIS
+  (math.fabs(self.rot1p)  -  math.fabs(rot1))  *  RREWARDANGLE),  (FACINGDIS_RANGE  -
tmp_par) * RREWARDFACINGANGLE * (math.fabs(self.rot2p) - math.fabs(rot2)))
                                    # print ("distance of past and this frame: " + str(self.distp) + " " +
str(dist1))
                                    # print ("angle difference of past and this frame: " + str(self.difap) + "
" + str(difa1))
                                    print ("Rewards: " + str(reward))
```

每走一次2米以内，不用按照最短路

按照最短路给（但是当时没到做这段代码）

```
                    self.lenp = dist1
                else:
                    myspfa  =  spfa(self.mp,  self.N,  self.M,  robot_global.position.x,
robot_global.position.z, rpos.x, rpos.z)
                    now_l, now_g_x, now_g_z = myspfa.getKey()
                    ang,      _,      _      =      getanglefea(robot_global.position.x,
robot_global.position.z, robot_global.rotation.w, now_g_x, now_g_z, 0)
                    reward  =  (self.lenp  -  now_l)  *  RREWARDDIS  +  (PI  /  2.0  -
math.fabs(ang)) * RREWARDANGLE
                    print("hjk--reward:   ", (self.lenp - now_l) * RREWARDDIS, (PI / 2.0 -
math.fabs(ang)) * RREWARDANGLE, reward)
                    self.lenp = now_l
                # time.sleep(10000000)
                actionout = open(actionoutPath, 'a+')
                print("action", action, "postionchange:", getDisXZ(robot_global.position.x,
robot_global.position.z,    0,    self.old_robot_postion.x,    self.old_robot_postion.z,    0),
"rotationchange:", np.fabs(robot_global.rotation.w - self.old_robot_rotation.w), file=actionout)

                sys.stdout.flush()
                actionout.close()
                self.old_robot_postion = copy.deepcopy(robot_global.position)
                self.old_robot_rotation = copy.deepcopy(robot_global.rotation)
                self.distp = dist1
                self.rot1p = rot1
                self.rot2p = rot2
                if dist1 < ARRIVEDIS and math.fabs(rot2) < ARRIVEANGLE:
                    reward = WINREWARD
                    print (dist1)

print("*******************************************************************************")

print("*******************************************************************************")

print("*******************************************************************************")

print("*******************************************************************************")

print("*******************************************************************************")

print("*******************************************************************************")
                    print("**********************************")
                    print("**********************************")
                    print ("*****OK! You shoot it!***********")
                    print("**********************************")
```

```
                                        print("***********************************")
                                        print("***********************************")

print("********************************************************************************")

print("********************************************************************************")

print("********************************************************************************")

print("********************************************************************************")

print("********************************************************************************")
                                        self.wintimes_win[self.init_dist_pos]                                    =
self.wintimes_win[self.init_dist_pos] + 1
                                        self.wintimes = self.wintimes + 1
                                        done = True
                                        #exit()



                print("*********** hjk-- already shot win for all episode:    ", self.wintimes, "collision:
", self.collisiontimes, self.wintimes_all, self.wintimes_win)
                return np.asarray(state), reward, done, {}

        def reset(self, testxml = 0, xml_human_x = -1, xml_human_y = -1, xml_human_rotation_z
= -1, xml_robot_x = -1, xml_robot_y = -1, xml_robot_rotation_z = -1):
                if testxml == 0:
                        for random_iter in range(20):
                                if self.my_case == -1:
                                        print("hjktest---- reset this", random_iter)

                                        randomrobotINT = random.randint(0, 9)
                                        if randomrobotINT == 0:
                                                robotx = 1.5
                                                roboty = 0.1
                                                robotz = 1.5
                                        elif randomrobotINT == 1:
                                                robotx = 2
                                                roboty = 0.1
                                                robotz = 1.5
                                        elif randomrobotINT == 2:
                                                robotx = 3
                                                roboty = 0.1
                                                robotz = 1.5
```

初始化
位置
（基本没用，
希望重写为改
在地图的任
[illegible]

```
    else:
        robotx = random.random() * (5.5 - 2) + 2
        roboty = 0.1
        robotz = random.random() * (8 - 4) + 4
    #robotx = random.random() * (5 - 1.5) + 1.5
    #roboty = 0.1
    #robotz = random.random() * (8 - 1.5) + 1.5
    '''
    robotx = random.random() * (5.5 - 3.8) + 3.8
    roboty = 0.1
    robotz = random.random() * (4.5 - 2.7) + 2.7
    '''
    robot_rotat_x = 0
    robot_rotat_y = 1
    robot_rotat_z = 0
    robot_rotat_w = random.uniform(-PI, PI)#-3.14, 3.14)    #-PI*4/5


    randomINT =random.randint(0, 4)
    if randomINT == 0:
        humanx = 3
        humany = 1.27
        humanz = 6
    elif randomINT == 1:
        humanx = 4.5
        humany = 1.27
        humanz = 7.5
    elif randomINT >= 2 and randomINT <= 4:
        humanx = 2.5
        humany = 1.27
        humanz = 2
    else:
        humanx = random.random()*(4 - 2.5) + 2.5
        humany = 1.27
        humanz = random.random()*(7.5 - 2.5) + 2.5
else:

    robotx = my_case_robotx[self.my_case]
    roboty = my_case_roboty[self.my_case]
    robotz = my_case_robotz[self.my_case]

    robot_rotat_x = my_case_robot_rotat_x[self.my_case]
    robot_rotat_y = my_case_robot_rotat_y[self.my_case]
    robot_rotat_z = my_case_robot_rotat_z[self.my_case]
```

```
                    robot_rotat_w = my_case_robot_rotat_w[self.my_case]

                    humanx = my_case_humanx[self.my_case]
                    humany = my_case_humany[self.my_case]
                    humanz = my_case_humanz[self.my_case]

                ###important!!!!!
                global global_rn
                global_rn = random.random()





                ### room
                position = Vector3()
                position.x = robotx
                position.y = roboty
                position.z = robotz





                rotation = Quaternion()
                rotation.x = robot_rotat_x
                #rotation.y = random.uniform(-3.14, 3.14)
                rotation.y = robot_rotat_y
                rotation.z = robot_rotat_z
                rotation.w = robot_rotat_w

                human_pose = Vector3()
                human_pose.x = humanx
                human_pose.y = humany
                human_pose.z = humanz

                rpos, rrot = getopposite(human_pose, human.rotation, self.my_case)
                dist1 = getDisXZ(position.x, position.z, rotation.w, rpos.x, rpos.z,
                                 rrot.w)
                distRtoH   =   getDisXZ(position.x,   position.z,   rotation.w,   human_pose.x,
human_pose.z, 0)
                if self.my_case != -1 or (dist1 < DISFROMSTARTTOEND and distRtoH >
MINDISFROMSTARTTOEND):
                    break
```

```
elif testxml == 1:
        position = Vector3()
        position.x = xml_robot_x
        position.y = 0.1
        position.z = xml_robot_y

        rotation = Quaternion()
        rotation.x = 0
        # rotation.y = random.uniform(-3.14, 3.14)
        rotation.y = 1
        rotation.z = 0
        rotation.w = xml_robot_rotation_z

        human_pose = Vector3()
        human_pose.x = xml_human_x
        human_pose.y = 1.27
        human_pose.z = xml_human_y
        global global_rn
        if xml_human_rotation_z == 1:
                global_rn = 0
        elif xml_human_rotation_z == 2:
                global_rn = 0.6
        elif xml_human_rotation_z == 3:
                global_rn = 0.2
        elif xml_human_rotation_z == 4:
                global_rn = 0.9



done = motor.set_velocity(0, 0, 0, 0)
while done is False:
        time_step.time_step_call()
        done = motor.set_velocity(0, 0, 0, 0)

pub_reset_node_physics_req()
while robot_global.reset_node_physics_res_flag is False:
        time_step.time_step_call()

pub_set_position_req(position)
while robot_global.set_position_res_flag is False:
        time_step.time_step_call()

pub_set_rotation_req(rotation)
while robot_global.set_rotation_res_flag is False:
```

```
            time_step.time_step_call()

        pub_get_human_position_req()
        while human.get_position_res_flag is False:
            time_step.time_step_call()
        pub_get_human_rotation_req()
        while human.get_rotation_res_flag is False:
            time_step.time_step_call()

        #print('hjk--- not set random :', human.position.x, human.position.y, human.position.z, human.rotation.x, human.rotation.y, human.rotation.z, human.rotation.w)

        pub_set_human_pose_req(human_pose)
        print("hjk--lll :", human.set_human_pose_res_flag)
        while human.set_human_pose_res_flag is False:
            time_step.time_step_call()

        pub_get_human_position_req()
        while human.get_position_res_flag is False:
            time_step.time_step_call()
        pub_get_human_rotation_req()
        while human.get_rotation_res_flag is False:
            time_step.time_step_call()

        print('hjk--- have set random :', human.position.x, human.position.y, human.position.z)

        laser_data, done = laser.get_laser_scan_data()
        while done is False:
            laser_data, done = laser.get_laser_scan_data()
            time_step.time_step_call()

        laser_state, is_collision = discretize_observation(laser_data, self.laser_dim, self.collision_threshold)



        for i in range(0, 10):
            time_step.time_step_call()

        ###############

        self.action_history1 = 0
        self.action_history2 = 0
        self.action_history3 = 0



        #get human pose
        pub_get_human_position_req()
        while human.get_position_res_flag is False:
                time_step.time_step_call()
        pub_get_human_rotation_req()
        while human.get_rotation_res_flag is False:
                time_step.time_step_call()


        # self.mp = np.zeros((self.N,self.M))

        rpos, rrot = getopposite(human.position, human.rotation, self.my_case)
        dist1 = getDisXZ(robot_global.position.x, robot_global.position.z, robot_global.rotation.w, rpos.x, rpos.z, rrot.w)


        rtang, rot1, rot2 = getanglefea(position.x, position.z, rotation.w, rpos.x, rpos.z, rrot.w)

        print("11111 state dis : ", dist1 * 10)
        state = changeStateFromEnvToNetwork(laser_state, dist1, rot1, rot2, -1)
        self.distp = dist1
        self.lenp = dist1
        self.rot1p = rot1
        self.rot2p = rot2
        # self.old_robot = robot_global
        self.old_robot_postion = copy.deepcopy(robot_global.position)
        self.old_robot_rotation = copy.deepcopy(robot_global.rotation)
        self.init_dist_pos = int(self.distp / 0.5)

        #self.init_dist_pos = self.init_dist / 0.5

        if self.init_dist_pos > self.wintimes_dist_key:
            self.init_dist_pos = self.wintimes_dist_key
        print("wwwww????? : ", self.init_dist_pos, self.distp)
        self.wintimes_all[self.init_dist_pos] = self.wintimes_all[self.init_dist_pos] + 1

        return np.asarray(state)
```