1. Descreva o código padrão Lua de comportamento Wander de um robô Pioneer p3dx. Descreva cada linha de código em suas palavras.

O código implementa o comportamento de navegação do tipo *Wander* para o robô Pioneer P3-DX no CoppeliaSim. Basicamente, ele faz o robô andar e desviar de obstáculos de forma automática. Abaixo eu explico o que acontece em cada parte do código.

O código começa com a chamada de uma função:

Função sysCall_init()

Essa é a primeira função que roda quando a simulação começa. Nela, o robô e os obstáculos são configurados.

A primeira linha da função pega a referência do robô na cena com:

local robot = sim.getObject('.')

Depois, ele cria uma coleção de objetos para os sensores detectarem:

local obstacles = sim.createCollection(0)

sim.addItemToCollection(obstacles, sim.handle_all, -1, 0)

sim.addItemToCollection(obstacles, sim.handle_tree, robot, 1)

Aqui, ele adiciona todos os objetos da cena na coleção de obstáculos, e depois remove o próprio robô dessa lista. Isso é importante porque senão o robô detectaria a si mesmo como obstáculo.

Em seguida, o código inicializa os 16 sensores ultrassônicos:

usensors = {}

for i = 1, 16, 1 do

usensors[i] = sim.getObject("./ultrasonicSensor", {index = i - 1})

sim.setObjectInt32Param(usensors[i], sim.proxintparam_entity_to_detect, obstacles)

end

Essa parte faz um laço para pegar cada sensor, de 1 a 16, e configurar para que ele detecte apenas os objetos da coleção de obstáculos (ou seja, tudo menos o próprio robô).

Depois ele pega os motores da esquerda e da direita:

motorLeft = sim.getObject("./leftMotor")

motorRight = sim.getObject("./rightMotor")

Logo em seguida, são definidos os parâmetros do comportamento de Braitenberg. São esses parâmetros que vão dizer como o robô reage aos obstáculos:

noDetectionDist = 0.5

maxDetectionDist = 0.2

Esses dois valores indicam a faixa de distância que os sensores vão considerar: se o obstáculo estiver a menos de 0.2, a influência dele é máxima. Se estiver a mais de 0.5, não influencia.

detect = {0, 0, ..., 0}

Aqui é criado um vetor com 16 zeros, onde cada posição vai armazenar o quanto cada sensor está detectando um obstáculo (de 0 a 1).

Depois temos dois vetores que representam o quanto cada sensor influencia a velocidade das rodas:

braitenbergL = {...}

braitenbergR = {...}

Cada posição desses vetores tem um peso que vai multiplicar o valor do sensor correspondente. Isso faz com que sensores da frente tenham mais influência para fazer o robô desviar.

Por fim, a velocidade base das rodas é definida e a função é encerrada:

v0 = 2

Após isso temos a segunda função:

Função sysCall_cleanup()

Essa função é chamada quando a simulação termina. No caso, ela está vazia porque não tem nada que precise ser limpo no final:

function sysCall_cleanup()

end

Logo após temos a chamada da terceira e última função, que descreve o'que deve ocorrer em cada passo da simulação:

Função sysCall_actuation()

Essa função roda o tempo todo enquanto a simulação está acontecendo. Ela atualiza o comportamento do robô.

O primeiro trecho é um for que lê os sensores:

```
for i = 1, 16, 1 do
    res, dist = sim.readProximitySensor(usensors[i])
```

Aqui ele verifica se cada sensor detectou algo (res > 0) e guarda a distância até o obstáculo (dist).

Depois ele calcula a influência de cada sensor:

```
if (res > 0) and (dist < noDetectionDist) then
    if (dist < maxDetectionDist) then
        dist = maxDetectionDist
    end
    detect[i] = 1 - ((dist - maxDetectionDist) / (noDetectionDist - maxDetectionDist))
else
    detect[i] = 0
end
```

Essa parte normaliza o valor da distância para um número entre 0 e 1, indicando o "quanto" o sensor está detectando algo. Se não estiver vendo nada, ele deixa o valor 0.

Depois disso, ele inicializa a velocidade das rodas:

```
vLeft = v0
vRight = v0
```

E agora entra a parte principal do Braitenberg:

```
for i = 1, 16, 1 do
    vLeft = vLeft + braitenbergL[i] * detect[i]
    vRight = vRight + braitenbergR[i] * detect[i]
end
```

Cada sensor altera a velocidade das rodas com base no seu peso e no quanto está detectando. Isso faz o robô virar para longe do obstáculo: se tem algo à esquerda, a roda direita desacelera e ele vira para a direita.

Por fim, o código define a velocidade final das rodas:

sim.setJointTargetVelocity(motorLeft, vLeft)

sim.setJointTargetVelocity(motorRight, vRight)

Assim essa função é encerrada e o código também. Em resumo o código faz o robô andar e desviar de obstáculos automaticamente, sem precisar de planejamento ou mapas. O robô apenas reage aos sensores com base em regras simples, mas que geram um comportamento bem natural. Esse tipo de lógica é inspirado no modelo de Braitenberg, e é uma forma bem eficiente de simular navegação básica com robôs móveis.

2. Descreva o código do robô Manta (controlável via teclado) em suas palavras.

O código implementa o controle de um carro no CoppeliaSim usando o teclado. Ele permite acelerar, frear e esterçar o carro usando as teclas direcionais. Abaixo está uma explicação detalhada de cada parte do código.

Função: if (sim_call_type==sim.syscb_init) then

Essa parte do código é executada no início da simulação. Nela, são feitas as configurações iniciais e a leitura dos componentes do carro.

steer_handle= sim.getObjectHandle('steer_joint')
motor_handle= sim.getObjectHandle('motor_joint')

Aqui o código pega as referências dos atuadores do carro: o volante (steer_joint) e o motor das rodas traseiras (motor_joint).

fl_brake_handle= sim.getObjectHandle('fl_brake_joint')
fr_brake_handle= sim.getObjectHandle('fr_brake_joint')
bl_brake_handle= sim.getObjectHandle('bl_brake_joint')
br_brake_handle= sim.getObjectHandle('br_brake_joint')

Essas linhas obtêm os "handles" (referências) dos freios das quatro rodas do carro: frente esquerda, frente direita, traseira esquerda e traseira direita.

max_steer_angle=0.5235987

Define o ângulo máximo que o volante pode girar, equivalente a 30 graus (em radianos).

motor_torque=60

Define a força máxima (torque) que o motor pode aplicar nas rodas.

dVel=1
dSteer=0.1

Define os incrementos de velocidade e de direção que serão aplicados ao pressionar as teclas.

steer_angle=0
motor_velocity=dVel*10
brake_force=0

Inicializa os valores de entrada: ângulo do volante começa em 0, velocidade inicial do motor é 10, e não há freio aplicado.

Função: if (sim_call_type==sim.syscb_actuation) then

Essa função é chamada a todo momento durante a simulação. Ela lê as entradas do teclado e atualiza os comandos do carro.

steer_pos=sim.getJointPosition(steer_handle)

Lê a posição atual do volante.

bl_wheel_velocity=sim.getObjectFloatParameter(bl_brake_handle,2012)
br_wheel_velocity=sim.getObjectFloatParameter(br_brake_handle,2012)
rear_wheel_velocity=(bl_wheel_velocity+br_wheel_velocity)/2
linear_velocity=rear_wheel_velocity*0.09

Aqui o código calcula a velocidade linear do carro com base na média das velocidades angulares das rodas traseiras, multiplicada pelo raio da roda (0.09 m).

Leitura do teclado:

message,auxiliaryData=sim.getSimulatorMessage()

Lê mensagens do simulador, como o pressionamento de teclas.

while message~=-1 do

Laço que processa todas as mensagens recebidas até que não haja mais nenhuma.

if (message==sim.message_keypress) then

Verifica se a mensagem recebida é de uma tecla sendo pressionada.

if (auxiliaryData[1]==2007) then

Se a tecla seta para cima for pressionada, aumenta a velocidade do motor.

if (auxiliaryData[1]==2008) then

Se a tecla seta para baixo for pressionada, reduz a velocidade (ou aplica freio se a velocidade for negativa).

if (auxiliaryData[1]==2009) then

Se a tecla seta para a esquerda for pressionada, gira o volante para a esquerda.

if (auxiliaryData[1]==2010) then

Se a tecla seta para a direita for pressionada, gira o volante para a direita.

if (auxiliaryData[1]==115) then

Se a tecla **S** for pressionada, a velocidade do motor é zerada (simula uma parada do veículo).

Controle de freio:

```
if (math.abs(motor_velocity)<dVel*0.1) then
   brake_force=100
else
   brake_force=0
end
```

Aplica força de frenagem nas rodas se a velocidade do motor estiver quase nula.

Limita o ângulo máximo de direção:

```
if (steer_angle> max_steer_angle) then
   steer_angle=max_steer_angle
end
if (steer_angle< -max_steer_angle) then
   steer_angle= -max_steer_angle
end
```

Garante que o volante não gire além do limite permitido.

Aplicação dos comandos ao carro:

sim.setJointTargetPosition(steer_handle, steer_angle)

Define a posição desejada do volante (ângulo de direção).

```
if(brake_force>0) then
   sim.setJointForce(motor_handle, 0)
else
   sim.setJointForce(motor_handle, motor_torque)
   sim.setJointTargetVelocity(motor_handle, motor_velocity)
end
```

Aplica torque ao motor se não estiver freando. Caso esteja freando, o torque do motor é zerado.

sim.setJointForce(fr_brake_handle, brake_force)
sim.setJointForce(fl_brake_handle, brake_force)
sim.setJointForce(bl_brake_handle, brake_force)
sim.setJointForce(br_brake_handle, brake_force)

Aplica a força de freio para todas as rodas do carro.

Função: if (sim_call_type==sim.syscb_cleanup) then

Essa função roda quando a simulação termina. No caso, está vazia, pois não há nada para limpar ao final da simulação.

Resumo
Esse código permite controlar um carro no CoppeliaSim usando o teclado. O usuário pode acelerar, frear e virar com as teclas direcionais. O controle é feito com incrementos suaves, respeitando os limites físicos do carro, como torque máximo e ângulo de direção. O sistema também aplica freios automaticamente quando a velocidade for muito baixa. Isso torna o controle simples, porém realista, simulando um carro de forma eficiente.