WGU C951

Task 2

Trayvonious Pendleton

Student ID # 011205284

10/24/2024

Part A

The chosen disaster recovery scenario simulates a lab environment that has been flooded. In this scenario, the robot must navigate through the flooded lab, where a desk, a chair, and another robot obstruct its path. The primary objective is for the robot to reach and detect a capsule located in the lab. These obstacles, including the desk, chair, and another robot, simulate the complex and unpredictable challenges faced during real-world disaster recovery missions.

Part B

The robotic recovery system is designed to significantly improve disaster recovery efforts within the flooded lab environment by navigating through obstacles to detect a critical capsule. The robot’s primary job is to autonomously find and detect the capsule, representing vital information or survivors in a real-world scenario. By completing this task, the robot provides valuable situational awareness in disaster recovery, ensuring that crucial areas are thoroughly searched and information is gathered.

After the desk, chair, and other robots are added as obstacles, the robot uses its enhanced navigation capabilities to maneuver around them, simulating real-world debris or equipment that could hinder rescue operations. The robot’s sensors enable it to detect obstacles and calculate alternative paths to reach its goal.

* **Desk and Chair:** These obstacles represent physical barriers the robot must avoid or move around. The robot’s proximity sensors help it detect and prevent collisions, ensuring it navigates efficiently through the cluttered environment without getting stuck or damaged.
* **Other Robot:** The presence of another robot introduces an unpredictable moving element. The recovery robot’s sensor suite helps it dynamically track the movements of this robot, adjusting its course in real-time to avoid collisions while continuing toward the capsule. This demonstrates the robot’s ability to handle dynamic and changing environments, which is crucial for disaster recovery operations where conditions can shift rapidly.

By overcoming these obstacles, the robot improves disaster recovery by reaching critical areas even when the environment is chaotic, flooded, or filled with debris. This capability allows first responders to focus on human efforts while relying on the robot to handle high-risk, hard-to-reach areas.

Part C

The robotic recovery system is built upon modifications to the base model, explicitly enhancing its ability to navigate the disaster recovery environment and detect critical objects, such as the capsule. Two key sensors were added to the robot’s architecture to improve its situational awareness and object detection capabilities: a **vision sensor** and a **proximity sensor**.

1. **Vision Sensor:**  
   The vision sensor was added to give the robot a more detailed understanding of its environment. This sensor allows the robot to scan the area visually, identifying obstacles such as the desk, chair, and another robot and detecting the flooded conditions. By processing the visual input, the robot can make informed decisions about its navigation path and avoid obstacles that could impede its progress. This capability is crucial in a disaster recovery scenario where the environment constantly changes or contains unpredictable hazards. The vision sensor’s data ensures that the robot is always aware of its surroundings, improving its ability to adapt and move toward its goal.
2. **Proximity Sensor:**  
   The proximity sensor was added to precisely detect the presence of the capsule, which represents a critical object in the disaster recovery scenario. The sensor is programmed to trigger when the robot approaches the capsule, turning green and printing "Capsule detected" to the console. This real-time feedback ensures that the robot can confirm the completion of its mission. The proximity sensor is essential in helping the robot determine when it has reached its destination despite any obstacles in its path. The robot can precisely locate critical objects in the environment by relying on proximity-based detection, even if the vision sensor is partially obstructed.

These modifications enhance the robot's ability to complete its task efficiently. The combination of the vision and proximity sensors allows the robot to navigate the environment precisely, avoid obstacles, and accurately detect the capsule, thus aiding the disaster recovery effort. Integrating these two sensors enables the robot to operate autonomously and provide valuable assistance in critical recovery missions.

Part D

The robot maintains an internal representation of the environment through the continuous input collected from its sensors. While the robot does not store a complete, detailed map of the environment in memory, it dynamically gathers information through its vision and proximity sensors to make real-time navigation and object detection decisions.

* **Vision Sensor Input:** The robot’s vision sensor provides a live feed of the surroundings, allowing it to "see" obstacles such as the desk, chair, and another robot. This sensor helps the robot detect the presence and position of obstacles in its path, allowing it to adjust its movements to avoid collisions. The visual data is processed in real time, guiding the robot through the environment. However, because this sensor only gathers immediate visual information, the robot does not create a permanent map of the environment. As a result, the robot must rely on constant sensor feedback to make decisions as it progresses through the area.
* **Proximity Sensor Input:** On the other hand, the proximity sensor is primarily used to detect when the robot is close to the capsule. This sensor doesn't provide continuous data about the overall environment but offers specific, vital information when the capsule is within range. When triggered, the sensor confirms to the console that the capsule has been detected and the goal has been reached.

The robot collects enough information through these sensors to build a temporary understanding of the environment at any given moment. While it doesn’t store detailed data or create a persistent internal map, it can effectively make decisions based on real-time inputs. This allows the robot to adapt quickly to environmental changes, such as shifting debris or dynamic obstacles like the other robot.

The main shortcoming of this system is the need for long-term memory or map-building capabilities. Since the robot doesn’t retain information from previous sensor readings, it has to continuously re-evaluate its surroundings, which may limit its efficiency in highly complex or more extensive environments. Despite this, the robot’s real-time decision-making allows it to perform well in controlled, cluttered environments like the flooded lab, where it can rely on live sensor data to reach its target.

Part E

The robot’s success in achieving its goal of detecting the capsule in the flooded environment is based on its ability to integrate several key concepts: knowledge, reasoning, uncertainty, and intelligence.

1. **Knowledge:**  
   The robot gathers knowledge about its environment using its vision and proximity sensors. The vision sensor provides real-time information about the surroundings, detecting obstacles such as desks, chairs, and other robots. The proximity sensor is designed to detect when the robot is near the capsule, signaling the robot’s objective. This constant flow of sensor data allows the robot to maintain a current understanding of its environment, helping it decide where to move and what obstacles to avoid.
2. **Reasoning:**  
   The robot makes decisions by processing the knowledge it has collected. As it receives input from the sensors, it reasons about the best course of action to achieve its goal. For instance, when the vision sensor identifies an obstacle, like a desk or chair, the robot must adjust its path to avoid a collision. Similarly, when the proximity sensor detects the capsule, the robot stops its search and confirms the completion of its mission. This real-time reasoning enables the robot to navigate the environment dynamically.
3. **Uncertainty:**  
   The robot must operate in an environment it does not know, adapting to changes and unexpected obstacles. Since it doesn’t store a predefined map or maintain the memory of the environment, the robot must deal with uncertainty by constantly reacting to new information as it navigates. The vision sensor helps the robot identify previously unknown obstacles, while the proximity sensor allows it to adapt to new conditions by confirming the capsule's location. The ability to adjust its behavior in real-time without relying on prior knowledge is crucial to the robot’s ability to handle uncertainty.
4. **Intelligence:**  
   The robot’s intelligence is demonstrated through its effective use of knowledge and reasoning to overcome uncertainty and achieve its goal. By collecting sensor data (knowledge) and making decisions based on that data (reasoning), the robot intelligently navigates the flooded environment, avoiding obstacles and finding the capsule. Its ability to adapt to the environment and make real-time decisions reflects a form of artificial intelligence critical for disaster recovery scenarios. The robot's intelligence is further shown by its ability to prioritize tasks—such as avoiding the moving robot—while focusing on reaching the final goal of detecting the capsule.

In summary, the robot integrates knowledge, reasoning, uncertainty, and intelligence to navigate the unpredictable environment and complete its mission. The robot can adapt to new situations and achieve its disaster recovery objectives by constantly gathering and interpreting sensor data.

Part F

While the current prototype effectively navigates a predefined environment and detects the capsule, several ways exist to improve its performance, particularly by incorporating reinforced learning and advanced search algorithms.

1. **Reinforced Learning:** Reinforced learning is a machine learning technique that allows the robot to learn from its environment through trial and error, receiving rewards or penalties based on its actions. In the current system, the robot navigates based on pre-programmed behaviors that respond to real-time sensor inputs. However, the robot could improve its decision-making process over time by introducing reinforced learning.

For instance, in a dynamic environment where the obstacles change locations or new ones are introduced, the robot could learn the most efficient paths through repeated attempts. Each time the robot successfully avoids obstacles and reaches the capsule, it could receive a "reward." If the robot collides with an obstacle or takes a longer path, it will receive a "penalty." Over time, these rewards and penalties would allow the robot to adjust its behavior, optimize its navigation strategy, and become more efficient at completing its tasks.

This learning process could make the robot more adaptable to varying conditions, such as changing debris patterns or moving obstacles like other robots, thus enhancing its usefulness in real-world disaster recovery scenarios.

1. **Advanced Search Algorithms:** Currently, the robot relies on simple goal-seeking behaviors to reach the capsule by avoiding obstacles as they are detected. However, this method might only sometimes be the most efficient, especially in more complex environments where there may be multiple possible paths or dead ends. Incorporating advanced search algorithms, such as *A (A-star)*\* or **Dijkstra's algorithm**, could significantly improve the robot’s navigation capabilities.
   * *A Search Algorithm:*\* A\* is an informed search algorithm that uses heuristics to find the shortest path to the goal. By using A\*, the robot could plan an optimal route through the environment before starting its movement rather than reacting to obstacles as they are encountered. This would help the robot avoid unnecessary detours and quickly find the most direct path to the capsule, even in highly complex or cluttered environments.
   * **Dijkstra's Algorithm:** This algorithm finds the shortest path in a weighted graph, which could be helpful if different parts of the environment have varying levels of difficulty to navigate. For example, some areas may be more flooded or cluttered with debris, and Dijkstra’s algorithm could help the robot prioritize more accessible paths that avoid high-risk zones.

By integrating reinforced learning and advanced search algorithms, the robot’s performance would improve over time, making it more efficient and adaptable in various disaster recovery situations. Reinforced learning would allow the robot to refine its actions through experience, while advanced search algorithms would optimize its pathfinding, ensuring faster and more reliable navigation toward its goal.

Part G

function sysCall\_init()

num\_To\_Detect = 0

noseSensorToDetect=sim.getObject("./sensingNoseToDetect") -- Handle of the proximity sensor

-- This is executed exactly once, the first time this script is executed

bubbleRobBase=sim.getObject('.') -- this is bubbleRob's handle

leftMotor=sim.getObject("./leftMotor") -- Handle of the left motor

rightMotor=sim.getObject("./rightMotor") -- Handle of the right motor

noseSensor=sim.getObject("./sensingNose") -- Handle of the proximity sensor

minMaxSpeed={50\*math.pi/180,300\*math.pi/180} -- Min and max speeds for each motor

backUntilTime=-1 -- Tells whether bubbleRob is in forward or backward mode

robotCollection=sim.createCollection(0)

sim.addItemToCollection(robotCollection,sim.handle\_tree,bubbleRobBase,0)

distanceSegment=sim.addDrawingObject(sim.drawing\_lines,4,0,-1,1,{0,1,0})

robotTrace=sim.addDrawingObject(sim.drawing\_linestrip+sim.drawing\_cyclic,2,0,-1,200,{1,1,0},nil,nil,{1,1,0})

graph=sim.getObject('./graph')

distStream=sim.addGraphStream(graph,'bubbleRob clearance','m',0,{1,0,0})

-- Create the custom UI:

xml = '<ui title="'..sim.getObjectAlias(bubbleRobBase,1)..' speed" closeable="false" resizeable="false" activate="false">'..[[

<hslider minimum="0" maximum="100" on-change="speedChange\_callback" id="1"/>

<label text="" style="\* {margin-left: 300px;}"/>

</ui>

]]

ui=simUI.create(xml)

speed=(minMaxSpeed[1]+minMaxSpeed[2])\*0.5

simUI.setSliderValue(ui,1,100\*(speed-minMaxSpeed[1])/(minMaxSpeed[2]-minMaxSpeed[1]))

end

function sysCall\_sensing()

local result,distData=sim.checkDistance(robotCollection,sim.handle\_all)

if result>0 then

sim.addDrawingObjectItem(distanceSegment,nil)

sim.addDrawingObjectItem(distanceSegment,distData)

sim.setGraphStreamValue(graph,distStream,distData[7])

end

local p=sim.getObjectPosition(bubbleRobBase,-1)

sim.addDrawingObjectItem(robotTrace,p)

end

function speedChange\_callback(ui,id,newVal)

speed=minMaxSpeed[1]+(minMaxSpeed[2]-minMaxSpeed[1])\*newVal/100

end

function sysCall\_actuation()

result=sim.readProximitySensor(noseSensor) -- Read the proximity sensor

-- If we detected something, we set the backward mode:

if (result>0) then backUntilTime=sim.getSimulationTime()+4 end

if (backUntilTime<sim.getSimulationTime()) then

-- When in forward mode, we simply move forward at the desired speed

sim.setJointTargetVelocity(leftMotor,speed)

sim.setJointTargetVelocity(rightMotor,speed)

else

-- When in backward mode, we simply backup in a curve at reduced speed

sim.setJointTargetVelocity(leftMotor,-speed/2)

sim.setJointTargetVelocity(rightMotor,-speed/8)

end

local result\_To\_Detect,distance,detectedPoint,detectedObjectHandle=sim.readProximitySensor(noseSensorToDetect)

if (result\_To\_Detect>0) then

if detectedObjectHandle then

if sim.getObjectAlias(detectedObjectHandle) == 'Capsule' then

num\_To\_Detect = num\_To\_Detect + 1

print("Capsule we want is detected!: Num: " .. tostring(num\_To\_Detect) .." - " .. tostring(sim.getObjectAlias(detectedObjectHandle)))

-- simInt simGetObjectColor(simInt objectHandle,simInt index,simInt colorComponent,simFloat\* rgbData)

-- Colors: Red: {1,0,0}, Green = {0,1,0}, Blue = {0,0,1}

sim.setObjectColor(noseSensorToDetect,0,sim.colorcomponent\_ambient\_diffuse,{0,1,0}) -- Green = {0,1,0}

end

end

else

sim.setObjectColor(noseSensorToDetect,0,sim.colorcomponent\_ambient\_diffuse,{0,0,1}) -- Blue = {0,0,1}

end

end

function sysCall\_cleanup()

simUI.destroy(ui)

end

Part I

No sources were used.