DISASTER RELIEF ROBOT

John McGinnes

03/09/2025

A. For this task, I chose an earthquake disaster recovery scenario set in a hotel room. The room has significant debris and the DisasterBot's mission is to search and rescue any survivors as it may be dangerous for human rescue crews to navigate. I've added four obstacles to the environment: a mattress, two nightstand tables, and a bureau, all of which are spread

across the room due to the earthquake. The obstacles will impede the robot's movement and require it to navigate around them while searching for survivors.

B. The robot improves disaster recovery by autonomously navigating the room, detecting obstacles, and avoiding collisions. Once the robot detects a survivor, it will stop, print "Survivor Located!" and pick up the person (carefully). It will then navigate back out of the room to the exit, avoiding debris and rescuing the survivor with no human intervention. This capability allows for faster and more efficient survivor identification and retrieval in hazardous environments without putting additional lives at risk in the process.

C. I modified the robot's architecture to include a proximity sensor (nose sensor) for obstacle detection and a custom proximity sensor for detecting the survivor ("Person_To_Detect"). The first sensor ensures that the robot can avoid obstacles like the mattress and furniture, while the second sensor detects the survivor, triggers the robot to stop, and prints "Survivor Detected!" before the robot picks up the survivor. These sensors are necessary to allow the robot to navigate the environment and perform rescue operations autonomously, especially in situations where human presence may be limited.

D. The robot maintains an internal representation of its environment using its proximity sensors to detect obstacles and the survivor. The robot's movement is based on feedback from the sensors, which continuously update its perception of the environment. It uses real-time data to avoid obstacles, and the detection of the survivor is the robot's goal. The robot's internal state is also tracked by variables like "num_To_Detect" and flags like "survivorDetected", which will change its behavior based on the situation.

E.

- Reasoning: The robot uses sensor data to determine its actions. If an obstacle is detected, it adjusts its movement to avoid collision. When the survivor is detected, the robot stops and changes the state to pick up the survivor. The reasoning is directed by the goal of detecting the survivor and it uses real-time data to accomplish that goal.
- Knowledge Representation: The robot's knowledge of the environment is represented through proximity sensor data, obstacle positions, and survivor location. The robot's behavior is driven by this knowledge, with changes in position and state being dependent on detected objects (either the survivor or obstacles).
- Uncertainty: The robot begins with uncertainty due to the limited starting knowledge of the positions of the obstacles and the survivor within the environment. It resolves that uncertainty using the proximity sensors to detect objects and adapt its actions based on input data.
- Intelligence: The robot demonstrates intelligence by making decisions based on sensor feedback. It will process input data, either obstacles or the survivor, to make decisions like stopping, moving around obstacles, or picking up the survivor.

F. The DisasterBot prototype could be improved by adding reinforced learning, allowing the robot to learn from trial and error in the environment to improve navigation and survivor detection. Over time, this would allow the robot to refine its behavior based on past data and optimize its actions for obstacle avoidance and survivor detection. Advanced search algorithms could also be used to help the robot plan better paths through the obstacles, like A* or Djkstra's Algorithm. This would also minimize travel time to get to the survivor. Communications could also be improved to provide the robot with a two-way radio, allowing the bot and survivor to communicate with emergency responders.

G. Python code for the robot is listed in full below:

```
import math
```

```
def sysCall_init():
    sim = require('sim')
    simUI = require('simUI')


    self.num_To_Detect = 0
    self.noseSensor_To_Detect =
sim.getObject("/bubbleRob/sensingNose_To_Detect")
    self.bubbleRobBase = sim.getObject('.')
    self.leftMotor = sim.getObject("/bubbleRob/leftMotor")
    self.rightMotor = sim.getObject("/bubbleRob/rightMotor")
    self.noseSensor = sim.getObject("/bubbleRob/sensingNose")
    self.minMaxSpeed = [50 * math.pi / 180, 300 * math.pi / 180]
    self.backUntilTime = -1
    self.robotCollection = sim.createCollection(0)
    sim.addItemToCollection(self.robotCollection, sim.handle_tree,
self.bubbleRobBase, 0)
    self.distanceSegment = sim.addDrawingObject(sim.drawing_lines, 4, 0, -
1, 1, [0, 1, 0])
    self.robotTrace = sim.addDrawingObject(sim.drawing_linestrip +
sim.drawing_cyclic, 2, 0, -1, 200, [1, 1, 0], None, None, [1, 1, 0])
    self.graph = sim.getObject('/bubbleRob/graph')
    self.distStream = sim.addGraphStream(self.graph, 'bubbleRob
clearance', 'm', 0, [1, 0, 0])


    # Create the custom UI
    xml = '<ui title="' + sim.getObjectAlias(self.bubbleRobBase, 1) + ' speed"
closeable="false" resizeable="false" activate="false">'
```

```
    xml = xml + '<hslider minimum="0" maximum="100" on-
change="speedChange_callback" id="1"/>'
    xml = xml + '<label text="" style="* {margin-left: 300px;}"/></ui>'
    self.ui = simUI.create(xml)
    self.speed = (self.minMaxSpeed[0] + self.minMaxSpeed[1]) * 0.5
    simUI.setSliderValue(self.ui, 1, 100 * (self.speed - self.minMaxSpeed[0])
/ (self.minMaxSpeed[1] - self.minMaxSpeed[0]))

    # Flag to track if the survivor has been detected
    self.survivorDetected = False
    self.personToDetect = None

def sysCall_actuation():
    result, *_ = sim.readProximitySensor(self.noseSensor)
    if result > 0:
        self.backUntilTime = sim.getSimulationTime() + 4
    if self.backUntilTime < sim.getSimulationTime():

        sim.setJointTargetVelocity(self.leftMotor, self.speed)
        sim.setJointTargetVelocity(self.rightMotor, self.speed)
    else:
        sim.setJointTargetVelocity(self.leftMotor, -self.speed / 2)
        sim.setJointTargetVelocity(self.rightMotor, -self.speed / 8)


    result_To_Detect, distance, detectedPoint, detectedObjectHandle, _ =
sim.readProximitySensor(self.noseSensor_To_Detect)
```

```
    if result_To_Detect > 0:
        if detectedObjectHandle:
            if sim.getObjectAlias(detectedObjectHandle) ==
'Person_To_Detect':
                self.num_To_Detect = self.num_To_Detect + 1
                if not self.survivorDetected:
                    print("Survivor located!")
                    self.survivorDetected = True  printing
                sim.setObjectColor(self.noseSensor_To_Detect, 0,
sim.colorcomponent_ambient_diffuse, [0, 1, 0])
                # Stop the robot when the person is detected
                sim.setJointTargetVelocity(self.leftMotor, 0)
                sim.setJointTargetVelocity(self.rightMotor, 0)



                self.personToDetect = detectedObjectHandle


    else:
        sim.setObjectColor(self.noseSensor_To_Detect, 0,
sim.colorcomponent_ambient_diffuse, [0, 0, 1])



    if self.personToDetect:
        robot_position = sim.getObjectPosition(self.bubbleRobBase, -1)
        # Move the detected person to follow the robot
        follow_position = [robot_position[0], robot_position[1],
robot_position[2] + 0.5]
```

```
        sim.setObjectPosition(self.personToDetect, -1, follow_position)


def sysCall_sensing():
    result, distData, *_ = sim.checkDistance(self.robotCollection,
sim.handle_all)
    if result > 0:
        sim.addDrawingObjectItem(self.distanceSegment, None)
        sim.addDrawingObjectItem(self.distanceSegment, distData)
        sim.setGraphStreamValue(self.graph, self.distStream, distData[6])
    p = sim.getObjectPosition(self.bubbleRobBase)
    sim.addDrawingObjectItem(self.robotTrace, p)


def speedChange_callback(ui, id, newVal):
    self.speed = self.minMaxSpeed[0] + (self.minMaxSpeed[1] -
self.minMaxSpeed[0]) * newVal / 100


def sysCall_cleanup():
    simUI.destroy(self.ui)
```

I. No external sources were quoted, paraphrased, or summarized in this submission.