June 3, 2024

Task 2: Disaster Relief Robot

A. Disaster Recovery Environment

An explosion occurred in a building, causing structural damage that makes it unsafe for humans to enter. To mitigate this problem, a disaster recovery team sends in a robot to search for survivors. In the current environment the robot is searching, there are two obstacles (a table and a plant) in the room with a person lying unconscious under the table (represented by the cylinder).

B. How the Robot Will Improve Disaster Recovery After Adding Obstacles

With the ability to independently traverse through dangerous environments and locate survivors, the robot can improve the disaster recovery process without putting human rescuers at potentially unnecessary risk. The robot's ability to handle obstacles ensures a more efficient and timely response in the search for survivors, further enhancing the safety of the overall disaster recovery situation.

C. Justification of Modifications Made to Robot

The robot's combination of a Lidar sensor and a heat sensor enhances its capabilities for disaster recovery. The lidar sensor allows for mapping of the environment along with obstacle avoidance while the heat sensor detects potential signs of life, which is crucial for identifying survivors. These abilities not only alert human responders to the location of survivors but also enable them to plan rescue efforts more effectively while prioritizing the safety of human responders.

D. How the Robot Maintains an Internal Representation of the Environment

The robot maintains an internal representation of the environment by combining data from its sensors—mapping from the Lidar sensor and the signs of life detected from the heat sensor. This data is processed using simultaneous localization and mapping (SLAM), allowing the robot to create and update a detailed map of its surroundings while estimating its location. Being able to continuously update its internal representation allows the robot to be able to dynamically create routes, avoid obstacles, and locate survivors in disaster recovery situations.

E. Robot's Implementation of Four Concepts

a. Reasoning

The robot can simulate reasoning via analyzing sensor data and its internal environment representation. With this data, the robot can make decisions about what route to take and how to avoid obstacles. It uses algorithms and decision-making processes to infer relationships between changing environmental factors and determine the most effective route.

b. Knowledge Representation

The robot represents knowledge about the environment through structured data formats, including maps, sensor readings, and hazard information. It organizes this knowledge in a way that allows for efficient storage, retrieval, and use during decision-making processes which permits it to effectively perform required tasks in complex environments.

c. Uncertainty

The robot deals with uncertainty by using techniques such as online replanning and model predictive control (MPC) to constantly adjust its plans based on new information it receives from its sensors. Additionally, the robot employs information-gathering actions, such as guarded movements and coastal navigation, to actively obtain new data and reduce uncertainty in its understanding of the current state of the environment. This helps the robot navigate difficult situations and make more informed choices, even when data is unavailable.

d. Intelligence

The robot demonstrates intelligence by independently adapting its behavior and decision-making processes based on observations and environmental conditions. Also, the robot uses machine learning algorithms to analyze and interpret data collected from its sensors, allowing it to recognize patterns and make predictions. It continuously learns from its experiences and environmental interactions, improving its performance and efficiency.

F. Future Prototype Improvements

To improve the prototype, we can teach it through trial and error using reinforcement learning. Also, advanced search algorithms such as Monte Carlo Tree Search will allow the robot to plan its routes more efficiently and make more informed decisions in complex situations by exploring different options and gradually figuring out the most effective path. These trials will help the robot make more effective moves, find the safest routes, and locate survivors more efficiently.

G. Robot Code

```
/********************************************************************************
*    Title: BubbleRob Tutorial
*    Author: CoppeliaSim
*    Date: June 3, 2024
*    Code version: Python
*    Availability: https://manual.coppeliarobotics.com/index.html
*
********************************************************************************/
#python

import math

num_to_detect = 0
nose_sensor_to_detect = None

def sysCall_init():
  # This is executed exactly once, the first time this script is executed
  global nose_sensor_to_detect
  sim = require('sim')
  simUI = require('simUI')
  num_to_detect = 0
  nose_sensor_to_detect = sim.getObject("./sensingNose_Detects")
  self.bubbleRobBase = sim.getObject('.') # this is bubbleRob's handle
  self.leftMotor = sim.getObject("./leftMotor") # Handle of the left motor
  self.rightMotor = sim.getObject("./rightMotor") # Handle of the right motor
  self.noseSensor = sim.getObject("./sensingNose") # Handle of the proximity sensor
  self.minMaxSpeed = [50*math.pi/180, 300*math.pi/180] # Min and max speeds for each
motor
  self.backUntilTime = -1 # Tells whether bubbleRob is in forward or backward mode
  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('./graph')
  # sim.destroyGraphCurve(self.graph, -1)
  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]))

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_actuation():
  global num_to_detect
  global nose_sensor_to_detect
  result, *_ = sim.readProximitySensor(self.noseSensor) # Read the proximity sensor
  # If we detected something, we set the backward mode:
  if result > 0:
      self.backUntilTime = sim.getSimulationTime() + 4
  if self.backUntilTime < sim.getSimulationTime():
      # When in forward mode, we simply move forward at the desired speed
      sim.setJointTargetVelocity(self.leftMotor, self.speed)
      sim.setJointTargetVelocity(self.rightMotor, self.speed)
  else:
      # When in backward mode, we simply backup in a curve at reduced speed
      sim.setJointTargetVelocity(self.leftMotor, -self.speed / 2)
      sim.setJointTargetVelocity(self.rightMotor, -self.speed / 8)

  result_to_detect, distance, detected_point, detected_object_handle, surface_vector = 
sim.readProximitySensor(nose_sensor_to_detect)
  if result_to_detect > 0:
      if detected_object_handle:
          if sim.getObjectAlias(detected_object_handle) == 'Cylinder_To_Detect':
              num_to_detect += 1
              print("Target cylinder found!: Num: " + str(num_to_detect) + " - " + 
sim.getObjectAlias(detected_object_handle))
              sim.setObjectColor(nose_sensor_to_detect, 0, sim.colorcomponent_ambient_diffuse, 
[0.0, 1.0, 0.0])
  else:
      sim.setObjectColor(nose_sensor_to_detect, 0, sim.colorcomponent_ambient_diffuse, [0.0, 
0.0, 1.0])

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

H. Panopto Video Recording

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=b0e6802b-5a57-4a84-9389-b18401304cd7

I. Sources

No outside sources were used in this task. The code is modified from a template from CoppeliaSim.