# Disaster Recovery Environment Description

## Disaster Recovery Environment

## The simulated environment within CoppeliaSim is conceptualized as a dense urban setting immediately following a significant seismic event or an analogous disaster. This digital area is populated with a variety of obstructions emblematic of real-world search and rescue scenarios, including varied debris and compromised structural elements that introduce complexity to navigation and strategy. In this disaster, BubbleRob is deployed by first responders to assess fire threats to firemen and report known locations as well as search for individuals at risk and report their last known location. Reporting is transmitted via console output to a logging window observed by deployers of the BubbleRob. The locations provided are a set of coordinates which also include elevation relative to the ground.

## Obstacles Added

* Fire (Orange Objects): Simulated fires are placed at random locations, representing the common aftermath of urban disasters. They pose risks of spreading and can block paths, requiring immediate attention. BubbleRob’s 360-degree fire proximity radar, when sensing a fire, will send the last known location of the fire with a message stating: “[Alert] – Fire# has been spotted at (x, y, z)”. BubbleRob stores these unique fires and logs them once to preserve energy and for easier readability. BubbleRob, upon identifying a fire and its location, it will back away and go in a different direction to continue its search.
* Persons (Blue Objects): These are simulated individuals stranded in the disaster zone, representing survivors in need of assistance. BubbleRob’s 360-degree person proximity radar, when it registers an individual, will send the last know location of the person in distress with a message: “Person# has been found at (x, y, z)!”. BubbleRob will also send these locations once and will not repeat them to keep logging easily readable to first responders. Once a person has been found, BubbleRob will continue searching for other individuals within the vicinity. BubbleRob will only back away and go on a different path if the most recent person found is in the direct path of collision with BubbleRob.

# B. Robot's Role in Disaster Recovery

# In the simulated disaster, BubbleRob serves as an autonomous agent for reconnaissance and initial assessment. Deployed into the environment where safety and accessibility are compromised, this robot plays a critical role in the preliminary stages of the disaster response operation.

# Assessment of Fire Hazards

# BubbleRob is outfitted with an advanced fire proximity sensor system that enables it to detect and localize fire outbreaks within the simulated urban landscape. The onboard sensor array provides 360-degree coverage, allowing for early identification of fires, which is critical for the safety of both trapped individuals and incoming human rescue teams. Upon the detection of a fire, BubbleRob calculates and transmits the precise coordinates of the hazard to the command center. This information, relayed through console output, aids in strategizing firefighting efforts and plotting safe rescue routes.

# Search and Localization of Survivors

# The search for survivors is augmented by BubbleRob's person proximity sensor, engineered to discern the presence of individuals within the disaster zone. As BubbleRob navigates the dangerous terrain, it actively scans for signs of life. When a survivor is detected, the robot promptly communicates their location with high precision, ensuring that the rescue personnel can respond quickly. This function is crucial for prioritizing rescue missions and allocating resources efficiently.

# Navigation and Data Transmission

# BubbleRob’s pathfinding algorithms enable it to navigate autonomously through the environment, dynamically adjusting its trajectory to avoid obstacles and hazards. The robot's logic systems allow for decision-making, it can distinguish between a detected fire and a survivor, ensuring appropriate reactions to each. It ensures that all detections are logged succinctly—once and only once—to facilitate clarity in the high-stress scenario it simulates. The robot's actions, paired with its internal mapping capabilities, make it an asset, capable of delivering critical data to the human teams set to intervene.

# By using these capabilities, BubbleRob ensures the efficacy of disaster response strategies, contributing significantly to the overarching goal of minimizing human risk and maximizing the efficiency of the recovery operation.

# C. Modifications to Robot Architecture

# To adapt BubbleRob for disaster recovery, several modifications were made to its base architecture, enhancing its capability to operate in a high-risk environment.

## Added Sensors

# Person Proximity Sensor (personRadar): This sensor is specifically designed to detect the presence of individuals within the disaster zone. The personRadar's sensitivity allows BubbleRob to identify survivors even in cluttered or chaotic environments. This sensor operates with a 360-degree range, ensuring that no corner of the simulated environment is left unchecked. The information it gathers is critical for rescue teams, as it provides the exact coordinates of detected survivors.

# Fire Proximity Sensor (fireRadar): The fireRadar is calibrated to identify simulated fires in the environment. Its purpose is to help BubbleRob avoid hazardous areas and, more importantly, to relay fire locations to emergency response teams. This sensor also has a 360-degree detection range, ensuring that the robot can navigate away from danger effectively. Its role is crucial in preventing BubbleRob from inadvertently entering hazardous zones and in assisting rescue teams in planning their approach to the disaster area.

## Sensors' Role in Disaster Recovery

# The integration of these sensors plays a pivotal role in the overall functionality of BubbleRob during disaster recovery operations. They provide the following benefits:

# Locating Survivors for Rescue: The personRadar sensor enables BubbleRob to identify individuals in need of assistance. By logging the exact location of each detection, it assists rescue teams in prioritizing their efforts, allowing them to focus on areas where survivors have been found. This information can significantly reduce response times and improve the chances of successful rescues.

# Identifying Fires for Containment Efforts: The fireRadar sensor is essential for identifying fire hazards. It ensures that BubbleRob avoids areas that are too dangerous to navigate while also alerting response teams to the exact location of fire threats. This sensor helps in strategizing firefighting efforts and planning safe routes through the disaster zone.

# By combining these sensors, BubbleRob can navigate complex and hazardous environments while providing critical information to first responders. The sensors serve as the robot's "eyes," allowing it to perform essential disaster recovery functions with a high degree of autonomy and accuracy.

# D. Robot's Internal Representation of Environment

## Internal Representation

# BubbleRob constructs an internal representation of its environment by synthesizing data from its various sensors through the simulated disaster zone. This internal mapping allows BubbleRob to maintain a real-time understanding of its surroundings, aiding in its autonomous navigation and obstacle avoidance.

# The robot's internal representation involves several key elements:

# Sensor Feedback: BubbleRob collects data from its sensors, particularly the personRadar and fireRadar. This data provides BubbleRob with critical information about its proximity to survivors and fire hazards. The robot processes this sensor feedback to make informed decisions about its path and to avoid potential dangers.

# Location Memory: BubbleRob stores the coordinates of key points of interest, such as the last known locations of detected fires and survivors. This memory allows the robot to avoid these points in the future or to direct human rescuers to them. The internal representation enables BubbleRob to operate with a sense of "awareness," minimizing redundant or unsafe movements.

# These components combine to create a robust internal representation that guides BubbleRob's behavior within the disaster recovery environment. It uses this information to navigate efficiently, adapt to changing conditions, and focus on high-priority tasks like locating survivors and avoiding fires.

# The internal representation is not only crucial for BubbleRob's autonomous navigation but also plays a significant role in the broader disaster recovery effort. By maintaining an accurate internal map of the environment, BubbleRob can support human-led operations by providing valuable data and reducing the risk for first responders.

# E. Concepts Implementation

## Reasoning

# BubbleRob demonstrates reasoning by interpreting sensor inputs to determine its next course of action. When the robot receives data from its sensors, it processes this information to make decisions about moving forward, turning, or reversing. This reasoning capability allows BubbleRob to create an effective search pattern within the disaster recovery environment. The reasoning process involves weighing various factors, such as proximity to obstacles or detected hazards, and choosing the most appropriate response. This leads to a more efficient exploration of the environment, as the robot can adapt to new information and avoid potentially dangerous paths.

## Knowledge Representation

# BubbleRob maintains an internal map of its environment, which acts as its knowledge base. This internal representation includes the locations of obstacles, detected fires, and identified survivors. By recording these points of interest, BubbleRob can make informed decisions about navigation and direct rescue teams to key areas. The internal map evolves as the robot explores, allowing it to avoid redundant paths and focus on new areas. This knowledge representation is crucial for the robot's ability to perform its tasks efficiently and to contribute valuable information to the broader disaster recovery effort.

## Uncertainty

# In a disaster recovery environment, uncertainty is a given. BubbleRob addresses this uncertainty by utilizing probabilistic sensor readings. The robot's sensors are designed to detect obstacles and hazards with a certain level of probability, recognizing that absolute certainty is often unattainable in dynamic environments. To cope with this uncertainty, BubbleRob incorporates random elements into its movement patterns, ensuring thorough coverage of the search area. This approach reduces the risk of missing critical information and allows the robot to operate effectively even when sensor data is ambiguous or incomplete.

## Intelligence

# BubbleRob's intelligence is exhibited through its ability to navigate the environment autonomously. This autonomy is based on the robot's dynamic adaptation to sensor inputs, enabling it to adjust its behavior in real time. As it encounters new obstacles or hazards, BubbleRob can modify its path and approach, demonstrating a level of flexibility and responsiveness. The robot's intelligence is further evidenced by its ability to log and remember key points of interest, facilitating efficient navigation and contributing to the success of the disaster recovery mission.

# F. Further Improvements

## Reinforced Learning

Reinforced learning can play a significant role in BubbleRob's evolution, allowing it to adapt and improve its decision-making processes over time. By employing this machine learning approach, BubbleRob could accumulate knowledge from its experiences within the disaster recovery environment, leading to more efficient search patterns and better obstacle avoidance.

Reinforced learning operates on the principle of rewarding desirable behaviors and discouraging undesirable ones. In this context, BubbleRob would receive positive reinforcement for successful outcomes, such as finding survivors or safely avoiding fire hazards. This reward-based system can guide BubbleRob toward more effective strategies, minimizing errors, and increasing operational efficiency.

A reinforced learning framework could involve:

* State Representation: Defining the various states BubbleRob can be in during its exploration of the environment. These states might include 'searching,' 'avoiding,' or 'reporting,' with transitions triggered by sensor inputs or predefined conditions.
* Reward System: Creating a reward structure that incentivizes BubbleRob to achieve specific goals, such as avoiding fires, locating survivors, or successfully navigating through complex areas. This system would drive the robot's learning process and encourage optimal behaviors.

By integrating reinforced learning, BubbleRob could develop a more dynamic and adaptable approach to disaster recovery, learning from its actions and optimizing its performance with each mission.

## Advanced Search Algorithms

Incorporating advanced search algorithms can significantly enhance BubbleRob's ability to navigate complex environments and ensure thorough coverage. Algorithms like A\* and Rapidly exploring Random Trees (RRT) offer robust solutions for pathfinding and exploratory movement.

* A\* (pronounced "A-star") is a popular pathfinding algorithm used in various applications, from robotics to gaming. It employs a combination of heuristic estimates and cost-based pathfinding to determine the most efficient route between two points. By integrating A\*, BubbleRob could effectively navigate from its starting point to a target location while avoiding obstacles and hazards.
* RRT is a tree-based search algorithm that explores random paths through an environment. It can be particularly useful for navigating complex or unknown terrain, as it generates a variety of paths, increasing the likelihood of finding a viable route. By using RRT, BubbleRob can traverse the disaster recovery environment with greater adaptability, allowing it to explore multiple pathways and identify the most effective routes.

Advanced search algorithms could enable BubbleRob to operate with greater precision and reliability, improving its ability to navigate the disaster recovery environment while fulfilling its primary objectives of locating survivors and avoiding hazards.

# G. BubbleRob’s core code (can also be found [here](https://github.com/amnotme/DisasterReliefBot-CoppeliaSim/blob/main/sources/bubble_rob_core_logic.py))

1. # python

2. import math

3. import random

4.

5. # Function to create a custom UI slider for controlling robot speed

6. def create\_slider():

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

8. xml += '<hslider minimum="0" maximum="100" on-change="speedChange\_callback" id="1"/>'

9. xml += '<label text="" style="\* {margin-left: 300px;}"/></ui>'

10. return xml

11.

12. # This function is executed once at the start of the simulation

13. def sysCall\_init():

14. # Retrieve essential handles and initialize attributes

15. sim = require('sim')

16. simUI = require('simUI')

17. self.bubbleRobBase = sim.getObject('.') # Handle for BubbleRob

18. self.leftMotor = sim.getObject("./leftMotor") # Handle for left motor

19. self.rightMotor = sim.getObject("./rightMotor") # Handle for right motor

20. self.noseSensor = sim.getObject("./sensingNose") # Handle for proximity sensor

21. self.personSensor = sim.getObject("./personRadar") # Handle for person sensor

22. self.fireSensor = sim.getObject("./fireRadar") # Handle for fire sensor

23.

24. # Set of unique identifiers for detected people and fires to prevent relogging

25. self.detected\_people = set()

26. self.detected\_fires = set()

27.

28. # Speed configurations for the motors

29. self.minMaxSpeed = [50\*math.pi/180, 300\*math.pi/180]

30. self.backUntilTime = -1 # Time until which the robot should move backwards

31.

32. # Create a collection for the robot and add visual elements for debugging

33. self.robotCollection = sim.createCollection(0)

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

35. self.distanceSegment = sim.addDrawingObject(sim.drawing\_lines, 4, 0, -1, 1, [0, 1, 0])

36. self.robotTrace = sim.addDrawingObject(sim.drawing\_linestrip + sim.drawing\_cyclic, 2, 0, -1, 500, [1, 1, 0], None, None, [1, 1, 0])1.

37.

38. # Create and configure the custom UI for speed control

39. self.ui = simUI.create(create\_slider())

40. self.speed = (self.minMaxSpeed[0] + self.minMaxSpeed[1]) / 2

41. simUI.setSliderValue(self.ui, 1, 100 \* (self.speed - self.minMaxSpeed[0]) / (self.minMaxSpeed[1] - self.minMaxSpeed[0]))

42.

43. # Regularly called to update sensor readings and visualization

44. def sysCall\_sensing():

45. result, distData, \*\_ = sim.checkDistance(self.robotCollection, sim.handle\_all)

46. if result > 0:

47. # Update visual elements if an object is detected

48. sim.addDrawingObjectItem(self.distanceSegment, None)

49. sim.addDrawingObjectItem(self.distanceSegment, distData)

50.

51. # Update the robot's path trace

52. p = sim.getObjectPosition(self.bubbleRobBase)

53. sim.addDrawingObjectItem(self.robotTrace, p)

54.

55. # Callback for speed slider UI element changes

56. def speedChange\_callback(ui, id, newVal):

57. self.speed = self.minMaxSpeed[0] + (self.minMaxSpeed[1] - self.minMaxSpeed[0]) \* newVal / 100

58.

59. # Function to handle the collision avoidance behavior

60. def collision\_detection\_handler(left\_motor\_jitter, right\_motor\_jitter):

61. if self.backUntilTime < sim.getSimulationTime():

62. # Move forward at the desired speed

63. sim.setJointTargetVelocity(self.leftMotor, self.speed)

64. sim.setJointTargetVelocity(self.rightMotor, self.speed)

65. else:

66. # Backup in a curve at reduced speed when an obstacle is detected

67. sim.setJointTargetVelocity(self.leftMotor, -self.speed / left\_motor\_jitter)

68. sim.setJointTargetVelocity(self.rightMotor, -self.speed / right\_motor\_jitter)

69.

70.

71. # Function to handle the sensing nose

72. def sensing\_object\_handler(left\_motor\_jitter, right\_motor\_jitter):

73. # If we detected something, we set the backward mode:

74. result, \*\_ = sim.readProximitySensor(self.noseSensor)

75. if result > 0:

76. self.backUntilTime = sim.getSimulationTime() + 1

77.

78. collision\_detection\_handler(left\_motor\_jitter, right\_motor\_jitter)

79.

80. # Function to handle person detection

81. def sensing\_person\_handler():

82. def find\_person(object):

83. # Check if the detected object is a person

84. for person in range(0, 7):

85. if sim.getObjectAlias(object) == f'Person{person}':

86. return True

87. return False

88.

89. # Read the person sensor and handle detection

90. person\_result, distance, point, object, n = sim.readProximitySensor(self.personSensor)

91. if person\_result > 0 and object and find\_person(object):

92. alias\_name = sim.getObjectAlias(object)

93. if alias\_name not in self.detected\_people:

94. # Log the detection only once

95. print(f"{alias\_name} has been found at {round(point[0], 4), round(point[1], 4), round(point[2], 4)}")

96. self.detected\_people.add(alias\_name)

97.

98. # Function to handle fire detection

99. def sensing\_fire\_handler(left\_motor\_jitter, right\_motor\_jitter):

100. def sense\_fire(object):

101. # Check if the detected object is a fire

102. for fire in range(0, 5):

103. if sim.getObjectAlias(object) == f'Fire{fire}':

104. return True

105. return False

106.

107. # Read the fire sensor and handle detection

108. fire\_result, distance, point, object, n = sim.readProximitySensor(self.fireSensor)

109. if fire\_result > 0 and object and sense\_fire(object):

110. self.backUntilTime = sim.getSimulationTime() + 0.5

111. if sim.getObjectAlias(object) not in self.detected\_fires:

112. # Log the detection only once

113. alias\_name = sim.getObjectAlias(object)

114. print(f"[Alert] - {alias\_name} has been spotted at {round(point[0], 4), round(point[1], 4), round(point[2], 4)}")

115. self.detected\_fires.add(alias\_name)

116.

117. collision\_detection\_handler(left\_motor\_jitter, right\_motor\_jitter)

118.

119. # Regularly called to handle actuation based on sensor inputs

120. def sysCall\_actuation():

121. # Introduce randomness to the motor behavior to create a more exploratory movement pattern

122. left\_motor\_jitter = random.choice([ i for i in range(1, 11)]) / 10

123. right\_motor\_jitter = random.choice([ j for j in range(8, 64, 4)])

124.

125. # Call sensor handlers

126. sensing\_object\_handler(left\_motor\_jitter, right\_motor\_jitter)

127. sensing\_fire\_handler(left\_motor\_jitter, right\_motor\_jitter)

128. sensing\_person\_handler()

129.

130. # Called once when the simulation ends to clean up

131. def sysCall\_cleanup():

132. simUI.destroy(self.ui)

# I. Acknowledgements and citations

This project does not contain any content that requires citations or references from external sources.