Simulation of a Search and Rescue Robot: Domain

CW REPORT FOR 4CCS1IAI INTRODUCTION TO ARTIFICIAL INTELLIGENCE

By

Fernando Lee | 1631508
Christian Honkanen | 1631080
Vishnu Thirumalai | 1631454
Sakeen Zaman | 1631323

Introduction

This domain models a disaster scenario (such as the 2013 Savar building collapse in Bangladesh) and multiple Search and Rescue robots. The main goal was to model a robot that could locate victims trapped inside collapsed buildings, or anywhere that disasters causing unsafe areas for human rescue teams might occur. The robot could provide emergency medical relief by injecting trapped victims in delicate conditions with Saline to sustain them until they can be rescued. It would additionally mark their locations for extraction. On the other hand, if a victim did not need medical attention, it would simply mark them. The robot could navigate through obstacles or hazards in the area by interacting with them in several ways, such as drilling through them or by moving over them.

This simulation greatly simplifies the problem, by assuming every person and obstacle has a fixed position, and that all of these positions are known from the beginning. Such things would be dynamically determined via sensors on the actual robot. It also assumes all robots have a collective knowledge of which people have been marked and which obstacles have been cleared in the case of multiple robots, which isn't as large of an assumption. The planner's goal is to simply find the quickest path that visits and interacts with every victim.

Description

To see the entire domain, please refer to Appendix 1.

Types identified in domain:

- "robot" – The object that moves around and injects and/or locates the people.
- "person" – The victim that needs to be injected and/or marked.
- "obstacle" – The hazards that need to be successfully navigated to reach the victims.
- "interactable" – "person" and "obstacle" are instances of this. Robots interact with these, and can only interact with one at any time. Likewise, each interactable can only be interacted with by one robot at any time.
- "locatable" – Both "robot" and "interactable" are instances of this. Used for objects that require a location, which for the purpose of this domain is everything barring the locations themselves.
- "location" – The places where the objects, and people are located.

Predicates identified in domain:

- "(at ?l - locatable ?loc - location)" – location where the locatable can be found on the grid.
- "(adjacent ?l1 ?l2 - location)" – identifies two locations as next to each other, so a robot may move from l1 to l2 if l2 fulfills other required conditions.
- "(clear ?l1 - location)" – there are no obstacles in this location.
- "(canGoThrough ?o - obstacle)" – the robot can directly go through this obstacle.
- "(canBePushed ?o - obstacle)" – the robot can push this obstacle out of the way.
- "(canGoOver ?o - obstacle)" – the robot can go over this obstacle.
- "(canGoAround ?o - obstacle)" – the robot can go around this obstacle
- "(canDrillThrough ?o - obstacle)" – the robot can drill through this obstacle.
- "(notLoadBearing ?o - obstacle)" – this obstacle supports a load (such as the ceiling or further rubble), thus should not be drilled through or pushed aside by the robot

- "(notBeingInteractedWith ?i - interactable)" – This object is available for interaction by a robot. Ensures that a single object doesn't have multiple robots interacting with it at once
- "(noInjection ?p - person)" – specifies that a person needs only to be marked, and not injected.
- "(needsMark ?p - person)" – specifies that a person has not yet been marked
- "(marked ?p - person)" – specifies that a person has been marked (used to specify goal conditions)
- "(notBusy ?r - robot)" – used to specify that the robot is not doing another action

Functions identified in domain:

- "(drillTime ?o - obstacle)" – The time it takes for the robot to drill through an obstacle.
- "(goOntoTime ?o - obstacle)" – Time taken to move onto an obstacle.
- "(goAroundTime ?o - obstacle)" – Time taken to go around an obstacle.
- "(goIntoTime ?o - obstacle)" – The time it takes for the robot to move into an obstacle.
- "(pushTime ?o - obstacle)" – Time taken for the robot to push aside an obstacle
- "(injectionTime)" – Time taken to inject and mark a victim.
- "(markTime)" – Time taken to mark a victim.
- "(normalMove ?r - robot)" – Time taken for the robot to simply move from one space to another clear space

Durative actions identified in domain:

Please note that all actions require have their durations based on various functions(e.g the action "DRILL" for a specific obstacle has the duration "drillTime" for that obstacle). For the full details, please check Appendix 1.

- "MARK-PERSON" – Needs the parameters "location", "person" and "robot". The planner needs to make sure at the start that the robot is not busy, the person isn't being interacted with, both the person and the robot are at the same location throughout the duration, the person hasn't been marked, and doesn't need an injection. Once this action is carried out, the person will be marked.
- "INJECT-PERSON" –Identical to the MARK-PERSON, but doesn't include a check for the person not requiring an injection. At the end of the action, the person will also be injected.
- "MOVE" – Needs the parameters "robot" and two "location". The planner needs to make sure the robot is not busy, is in the first given location, the second given location is empty, and the two locations are adjacent. Once this action is carried out, then the robot will move to the second location.
- "DRILL" – Needs the parameters "robot", two "location" and an "obstacle. The planner needs to check that the robot is not busy, the obstacle isn't being interacted with, the obstacle isn't load bearing and can be drilled through, the robot is in the first location, and the obstacle is located in the second, adjacent location. Once this action is carried out, then the obstacle will be removed and the second location marked clear.
- "PUSH-ASIDE" – Identical to DRILL, but checks that the obstacle can be pushed aside rather than drilled.
- "MOVE-AROUND" - Needs the parameters "robot", three "location" and "obstacle". The planner needs to check that the robot is not busy, the obstacle isn't being interacted with, the robot is at the first location, the obstacle is at the second adjacent

location, and the third location is clear and adjacent to the second location. Once this action is carried out, the robot will move to the third location.

- "MOVE-INTO" – Needs the parameters "robot", two "location" and an "obstacle. The planner needs to check that the robot is not busy, the obstacle isn't being interacted with, the obstacle can be moved into, the robot is in the first location, and the obstacle is located in the second, adjacent location. It will then move the robot to the location of the obstacle.
- "MOVE-ONTO" – Identical to MOVE-INTO, but checks the obstacle can be moved onto rather than into.

Design Choices

The domain treats the area similar to a chessboard, with each space being equally adjacent to the ones above, below and to the left and right of it. This is as opposed to a graph/tree like structure, where each location is linked to several other locations via edges. While this 'chessboard' approach does mean there are several empty squares on the map, which wouldn't be present for the graph (they'd be part of the edges), this allows easier handling of obstacles, since an empty space will be left behind on their removal, and a constant move time between each location, which can then be affected by various factors etc. Since OPTIC doesn't allow the use of ADL, these factors have been directly added into the various times. e.g. goIntoTime is the normalMove+ the time due to the obstacle on the square, and injectionTime is markTime +the time for the injection alone.

Several predicates (such as noInjection) were included due to OPTIC not being able to handle negative preconditions. This is clearly shown in needsMark/marked, as one is needed as a precondition and the other to define the goal state. It's not required to have an 'injected' predicate, because MARK-PERSON isn't available to those victims without the 'noInjection' predicate, so the planner must use INJECT-PERSON on them to obtain 'marked'.

A conscious decision was made to not allow multiple robots to interact with the same object at once. While this could be rectifiable (reducing the time taken to drill an obstacle when two robots work on it, for example), not allowing it greatly simplifies the problem and prevents multiple failure cases, such as two robots colliding when passing through or onto an obstacle. One example of a fix could be to given a 'canDrillThrough' obstacle a durability and have the drill action take a fixed time (per robot) and reduce this durability on completion, marking it clear when the durability hits zero.

An obstacle can have multiple permissions attached to it (e.g a rock fall could be pushed aside or climbed over), and the planner would choose the fastest method of traversing it. For the given example, climbing over is normally faster, but multiple robots might benefit from the space being marked clear after being pushed aside.

Appendix 1: Domain file

```
(define (domain DisasterRobot)

(:requirements :strips :typing :durative-actions :fluents :equality)

(:types
       robot interactable - locatable
        person obstacle - interactable
        location
)

(:predicates
        (at ?l - locatable ?loc - location)
       (adjacent ?l1 ?l2 - location)
       (clear ?l1 - location)
        (canGoThrough ?o - obstacle)
        (canBePushed ?o - obstacle)
        (canGoOver ?o - obstacle)
        (canGoAround ?o - obstacle)
        (canDrillThrough ?o - obstacle)
        (notLoadBearing ?o - obstacle)
        (notBeingInteractedWith ?i - interactable)
        (noInjection ?p - person)
        (needsMark ?p - person)
        (marked ?p - person)
        (notBusy ?r - robot)
 )

(:functions
        (drillTime ?o - obstacle)
        (goOntoTime ?o - obstacle)
        (goAroundTime ?o - obstacle)
        (goIntoTime ?o - obstacle)
        (pushTime ?o - obstacle)
        (injectionTime)
        (markTime)
       (normalMove ?r - robot)
)



(:durative-action MARK-PERSON
       :parameters (?l - location ?p - person ?r - robot)
       :duration (= ?duration (markTime))
       :condition (and
             (at start (notBusy ?r))
             (at start (notBeingInteractedWith ?p))
             (at start (needsMark ?p))
             (at start (noInjection ?p))
             (over all (at ?p ?l))
             (over all (at ?r ?l)))
       :effect (and
             (at start (not(notBusy ?r)))
             (at start (not(notBeingInteractedWith ?p)))
             (at end (not(needsMark?p)))
             (at end (marked ?p))
             (at end (notBeingInteractedWith ?p))
             (at end (notBusy ?r))))

(:durative-action INJECT-PERSON
       :parameters (?l - location ?p - person ?r - robot)
       :duration (= ?duration(injectionTime))
       :condition (and
```

```
            (at start (notBusy ?r))
            (at start (notBeingInteractedWith ?p))
            (at start (needsMark?p))
            (over all (at ?p ?l))
            (over all (at ?r ?l)))
        :effect (and
            (at start (not(notBusy ?r)))
            (at start (not(notBeingInteractedWith ?p)))
            (at end (not(needsMark?p)))
            (at end (marked ?p))
            (at end (notBeingInteractedWith ?p))
            (at end (notBusy ?r))))

(:durative-action MOVE
    :parameters (?l1 ?l2 - location ?r - robot)
    :duration (= ?duration (normalMove ?r))
    :condition (and
        (at start (notBusy ?r))
        (at start (at ?r ?l1))
        (at start (adjacent ?l1 ?l2))
        (at start (clear ?l2)))
     :effect (and
        (at start(not(notBusy ?r)))
        (at end (not(at ?r ?l1)))
        (at end (at ?r ?l2))
        (at end (notBusy ?r))))

(:durative-action DRILL
    :parameters (?l1 ?l2 - location ?r - robot ?o - obstacle)
    :duration (= ?duration(drillTime ?o))
    :condition (and
        (at start (notBusy ?r))
        (at start (notBeingInteractedWith ?o))
        (over all (notLoadBearing ?o))
        (at start (at ?r ?l1))
        (at start (adjacent ?l1 ?l2))
        (at start (at ?o ?l2))
        (at start (canDrillThrough ?o)))
     :effect (and
        (at start(not(notBusy ?r)))
        (at start (not(notBeingInteractedWith ?o)))
        (at end (clear ?l2))
        (at end (notBeingInteractedWith ?o))
        (at end (notBusy ?r))))

(:durative-action PUSH-ASIDE
    :parameters (?l1 ?l2 - location ?r - robot ?o - obstacle)
    :duration (= ?duration(pushTime ?o))
    :condition (and
        (at start (notBusy ?r))
        (at start (notBeingInteractedWith ?o))
        (at start (at ?r ?l1))
        (at start (adjacent ?l1 ?l2))
        (at start (at ?o ?l2))
        (over all (notLoadBearing ?o))
        (at start (canBePushed ?o)))
     :effect (and
        (at start(not(notBusy ?r)))
        (at start (not(notBeingInteractedWith ?o)))
        (at end (clear ?l2))
        (at end (notBeingInteractedWith ?o))
        (at end (notBusy ?r))))

(:durative-action MOVE-AROUND
    :parameters (?l1 ?l2 ?l3 - location ?r - robot ?o - obstacle)
    :duration (= ?duration(goAroundTime ?o))
    :condition (and
```

```
            (at start (notBusy ?r))
            (at start (notBeingInteractedWith ?o))
            (at start (at ?r ?l1))
            (at start (adjacent ?l1 ?l2))
            (at start (adjacent ?l2 ?l3))
            (at start (at ?o ?l2))
            (at start (clear ?l3))
            (at start (canGoAround ?o)))
        :effect (and
            (at start(not(notBusy ?r)))
            (at start (not(notBeingInteractedWith ?o)))
            (at end (at ?r ?l3))
            (at end (not(at ?r ?l1)))
            (at end (notBeingInteractedWith ?o))
            (at end (notBusy ?r))))

(:durative-action MOVE-INTO
    :parameters (?l1 ?l2 - location ?r - robot ?o - obstacle)
    :duration (= ?duration(goIntoTime ?o))
    :condition (and
            (at start (notBusy ?r))
            (at start (notBeingInteractedWith ?o))
            (at start (at ?r ?l1))
            (at start (adjacent ?l1 ?l2))
            (at start (at ?o ?l2))
            (at start (canGoThrough ?o)))
        :effect (and
            (at start(not(notBusy ?r)))
            (at start (not(notBeingInteractedWith ?o)))
            (at end (at ?r ?l2))
            (at end (not(at ?r ?l1)))
            (at end (notBeingInteractedWith ?o))
            (at end (notBusy ?r))))

(:durative-action MOVE-ONTO
    :parameters (?l1 ?l2 - location ?r - robot ?o - obstacle)
    :duration (= ?duration(goOntoTime ?o))
    :condition (and
            (at start (notBusy ?r))
            (at start (notBeingInteractedWith ?o))
            (at start (at ?r ?l1))
            (at start (adjacent ?l1 ?l2))
            (at start (at ?o ?l2))
            (at start (canGoOver ?o)))
        :effect (and
            (at start(not(notBusy ?r)))
            (at start (not(notBeingInteractedWith ?o)))
            (at end (at ?r ?l2))
            (at end (not(at ?r ?l1)))
            (at end (notBeingInteractedWith ?o))
            (at end (notBusy ?r))))

)
```