Autonomous Building of Structures in Unstructured Environments via AI Planning

Jamie O. Roberts [1, 2*], **Santiago Franco** [3*], **Adam A. Stokes**[1], **Sara Bernardini** [3]
* [1]School of Engineering, Institute for Integrated Micro and Nano Systems, The University of Edinburgh
[2]EPSRC CDT in Robotics and Autonomous Systems, Edinburgh Centre for Robotics
[3]Department of Computer Science, Royal Holloway University of London

Abstract

In this paper, we offer a novel AI planning representation, based on a Cartesian coordinate system, for enabling the autonomous operations of Multi-Robot Systems in 3D environments. Each robot in the system has to conform to unique actuation and connection constraints that create a complex set of valid configurations. Our approach allows Multi-Robot Systems to self-assemble themselves into larger structures via AI planning, with the overarching goal of providing structural capabilities in harsh and uncertain environments.

In comparing four different PDDL (Planning Domain Definition Language) domain representations, we show that our novel formulation satisfies the practical requirements emerging from robot deployment in the real world, resulting in an AI planning system that is accurate and efficient. We scale up performance by implementing direct FDR (Finite Domain Representation) generation based on the best performing PDDL model, bypassing the PDDL-to-FDR translation used by the majority of modern planners. The proposed approach is general and can be applied to a broad range of AI problems involving reasoning in 3D spaces.

1 Introduction

We introduce a Multi-Robot System (MRS) that self-assembles into a usable structure and is specifically targeted for extreme environments. Our system is intended for use in one of the harshest environments that humans have to operate in currently, the high radiation zones found in nuclear reactors and particularly nuclear waste storage infrastructure.

The high levels of radiation in these zones establish a unique temporal constraint on electronic operations as they impose a hard time limit for them, severely reducing the effectiveness of robotics in these environments. In effect, the critical goal for any successful robotic deployment in such environments is the maximisation of the amount of time that the robots have to be actually doing their intended task. All other time is not only wasted from an operational task perspective, but also for the lifespan of the robot. Currently, robotic systems are unable to operate effectively in these environments because of the high uncertainty that characterises them. By deploying an MRS that assembles itself into a fixed structure, other service robots can use the structure as a known, obstacle-free map to maximise their time doing useful tasks.

The environments that these systems are intended to operate in impose strict requirements, namely accountability and verification in deploying autonomous systems. As AI Planning uses a symbolic, logic-based approach, it is particularly well-suited to tasks where constraints are clearly defined, the applicability of actions can be easily formulated and the solutions can be explained and are guaranteed to comply with the constraints. In nuclear decommissioning, human operators always supervise missions and plans and Task Planning, when appropriately displayed, represents a clear and intelligible means of communication between machines and humans.

Utilising the planning language PDDL 2.1 further adds to the suitability of the problem above as it allows for a rich goal language. It will not always be possible to know exactly the kind of structure that is needed at the start of the mission, but the human operator might just want to specify certain locations that must have part of the structure or even a part of the structure that is oriented in a specific way.

The flexibility that PDDL allows, using any literal as a potential goal, provides a major benefit.

In this paper, we offer a novel AI planning representation that enables an MRS to self-assemble in a 3D environment. In particular, we present a new formulation of planning domains that involve reasoning in 3D spaces based on a *canonical* representation of the space and a *countable* representation of the objects in it.

After discussing related work in Section 2, we explain the structure and functioning of the MRS in Section 3. In Section 4, we detail the formulation of a planning domain that represents the MRS and lends itself to efficient planning thanks to the introduction of new features in the modelling of 3D space and objects. We then exhibit the power of our representation by presenting a large set of experiments in Section 5. They show that our technique outperforms other suitable representations of the problem. Section 6 offers concluding remarks.

[*]These two authors contributed equally to this work.

2 Related Work

MRSs that self-assemble into useful structures have long been an attractive goal for academic research and industry alike. Currently, the academic field is largely focused on the physical hardware deployed to achieve self-building robot systems with most contributions in the field related to actuation and connection mechanisms that are robust and may simplify control strategies.

Whilst there are many MRSs, this section will be limited to MRS systems that were designed to self assemble into a structure. Romanishin et al. (2015) present the M-Blocks: magnetic agents that are able to locomote over each other to create cubic structures. The work focuses on the actuating methods for this robot.

Stewart et al. (2006) present a distributed feedback mechanism to construct a 'wall' via individual robot agents. Kotay et al. (1998) propose the self-reconfiguring robotic molecule and explore control mechanisms as opposed to the hardware of the individual agents. Jenett et al. (2019) propose a robotic system that constructs a 3D structure by carrying building blocks and arranging them. The authors report a control strategy based on simple rules concerning the location of the constructor robots and the structure it is currently located on.

In the field of AI Planning, there has been limited work in planning for problems with spatial dependencies. Efficient planning for structures in 3D has not yet been achieved, and planning domains for robotic applications are not typically focused on spatial reasoning about the robot environment, but rather on topological relationships between sub-task dependencies in order to achieve a larger goal.

In integrating PDDL (Fox and Long 2011) with robotic control strategies, Cashmore et al. (2015) present Rosplan, a package which allows for inline use of AI Planning in ROS. There has been work towards integrating AI Planning with robotic control problems. Munoz et al. (2010) present a PDDL-based planner for the Ptinto robot which demonstrates some reasoning performed on a 2D environment by the PDDL domain. Balakirsky et al. (2012), Dornhege et al. (2013), Estivill et al. (2013), Quintero et al. (2011), Leidner et al. (2013) and Toussaint et al. (2015) introduce spatial planning in work which combines both task and motion planning to be solved by optimizing over a final geometric state, as opposed to symbolic representations. This also means that the problems cannot be solved by all symbolic planners. Toussaint et al. (2015) and Leidner et al. (2013) all demonstrate PDDL being used to perform spatial reasoning in performing robotic tasks, but the spatial reasoning is abstract and of features such as objects 'on top' of other objects. Takahashi et al. (2017) also demonstrates spatial reasoning in order to produce structures from cubic blocks. In their work, the spatial reasoning is performed abstractly.

Wolter et al. (2015) present on planning with qualitative spatial relationships for a single robot, with the focus on learning unknown forward models using LTL and QSL.

Asl et al. (2014) present interesting work in qualitative calculus for spatial reasoning in heuristic search. Belouaer et al. (2011) present a study on spatial knowledge representations in planning languages but the work is focused around

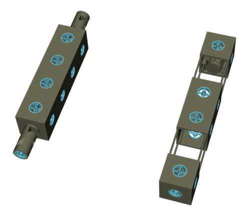
Figure 1: Representation of structural robot. *Left*: Robot with manipulators fully deployed. *Right*: Robot with manipulators retracted and main body extended for locomotion.

spatial qualities such as objects overlapping and in front of other objects. Churchill et al. (2011) present a PDDL domain which shows scheduling of building tasks in the video game Starcraft.

Reasoning in 3D spaces is hard to scale due to the large increase in computational effort as the environment grows. Planning techniques help becasue they rely on search algorithms that use efficient duplicate detection in the state space. However, they have problems in dealing with functionally equivalent search states (with regard to reaching the goal optimally, functionally equivalent states can be treated as duplicates). For example, for most problems in the well-known planning domain Gripper (Koehler 1998), the identity of the balls does not matter. States can be treated as duplicates if they have the same number of balls in each room because the goal only specifies how many balls are needed in each room. Functionally equivalent states are not automatically detected as duplicates by A* and its sub-optimal variants. Symmetry detecting strategies like *Baggy* (Riddle et al. 2015), which are based on transforming functionally equivalent predicates into a single 'countable' predicate, can increase planning performance exponentially. Alternatively, symmetries can also be pruned when detected in the search graph (Alkhazraji et al. 2014), albeit this method incurs inline computational costs.

Ultimately, the work towards spatial reasoning in AI Planning has not, so far, been performed on the robot level and spatial relationships have not been inferred from the 3D positioning of robots in their environments.

3 The Multi-Robot System

Structural Robots

The individual Structural Robots (SR) that constitute the MRS are designed to occupy a 'Manhattan Grid' representation of the environment in 3D. The robots are designed to self-assemble into cubic-like structures and so the robot has a discrete number of motions and configurations.

Figure 1 is a representation of the key features of the robot. For the purposes of the AI Planning, it can be considered as a uniform cuboid. Figure 1 shows four equally spaced connection points on each face of the robot. Connections can be made by gripping in between the blue connection points in Figure 1.

The locomotive ability of the robot is achieved by extending the end portions of the main body. It can move by independently actuating these sections and so it employs an 'inchworm' like motion.

The linkages at the ends of the robot are its manipulator mechanisms. Each robot has identical actuating manipulators which are able to connect to the blue connectors and so the SR is completely symmetrical. Figure 2 is a closer examination of the robot's manipulators. Each manipulator has 3 degrees of freedom, which are denoted by A, B and C in Figure 2. A and C rotate about the Y-axis, and B rotates about the X-axis. The manipulator serves to manipulate either the robot itself, or another robot connected to it.

As the manipulators can only be deployed from the ends of the robot, structural connections are achieved by manoeuvring each robot so that the manipulator agrees with spatial constraints. This creates the majority of the complexity in looking for valid actions. Added to these constraints, there are also structural considerations that must be considered in planning for viable structures.

Actions

An SR can move in 3D space only by attaching itself to other robots of the same type. Each robot gets power and structural support by being connected to another robot. One of the robots is assumed to be connected to a power source. Hence, a robot must be connected to another robot at all times. The MRS as a whole provides a 'path' upon which other robots can move along to reach previously unreachable areas.

The motions of the robots are described in terms of the orientations, initial and final positions of the designated origin point, which corresponds to a connection point. All action orientations are described using the positive angles and in terms of a displacement in each dimension (we use the x-axis in the examples). Any displacement between positions described is equivalent to the distance between the non-actuating connection points on the robot.

We selected the actions based on the actual locomotive capabilities of the robot, with the aim to be able to produce an orthogonal structure, in keeping with the goal of producing a lattice-like structure. One of the key features of this domain representation is that the actions are intuitive for human operators. The action formation is a modular process with the possibility of additions and removals depending on the case specific needs of the system before deployment. The actions described below are representative of a common set to form lattice structures.

The naming convention of the actions is structured around the MRS 'placing' a robot or 'traversing' a robot across itself, the principal distinction being that some actions place a robot in a location, which can then be used by other robots to move around the existing structure.

Figure 3 can be used as a reference to follow the explanations of all actions. The figure is split into A and B, initial and final states of the action, respectively. The annotation from one to four indicates the connection points, in ascending order from the reference connection point which is denoted by the axes going through the first connection point. In the following explanations, robot translations are shown

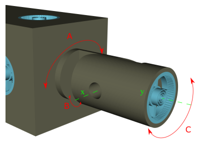

Figure 2: Detailed view of the robot manipulator with three degrees of freedom A, B and C in red, and the x and y axes about which they rotate in green. A and C both actuate about the y-axis but are denoted separately as they represent different points of actuation.

by displacements in a given axis, (e.g. +4 in the x). The displacement is in terms of connection points so, +4 in the x is the spacing of four connection points in the x dimension. Each action is described in terms of one set of displacements, but actions can have different displacements depending on the dimensions the actions happen in. The format in the explanations remains the same, each displacement acts on the reference coordinate as mentioned before.

Traverse The traverse action is the basic movement of the SR, allowing the robot to move on other robots. The result of this action is +1 in the x direction. There is no change in the y or z direction and no change in robot orientation. This action is employable in the x, y and z axis.

Place_line The place_line action allows the system to extend in one dimension and form a line. The result of this action is that the origin position extends by +4 in the x, and -1 in the z direction. There is no change in the y-axis and the orientation. This action allows a 'line' to be constructed and is employable in the x,y and z axis.

Place_right_angle The place_right_angle action (Figure 3) is employed to form a right-angle in the structure. The result of this motion is +4 points in the x, -1 in the z direction and -3 in the y direction. The difference between this action and the place_line action is that in place_right_angle the final orientation of the robot has changed by 90 degrees. This action is only employable in the xy plane.

Traverse_right_angle The action traverse_right_angle is used by an SR to manoeuvre around a right angle on an existing structure. This action results in a right-angle change in the robot positions. The result of this action is +4 in the x, +1 in the y and no change in the z direction. The robot orientation also changes by 90 degrees. This action can only be employed in the xy plane.

Traverse_vertical The action traverse_vertical is primarily used to build the structure into the z direction. The result of this motion is +4 in the x and no change in the y or z direction. The change in orientation is always in the z direction for this action and can only be employed from the xy plane and into the z axis.

Traverse_horizontal The action traverse_horizontal allows a vertical robot to be returned to a horizontal position. The result of this action is +1 in the x, and no change in the y or z directions, with an orientation change of 90 degrees

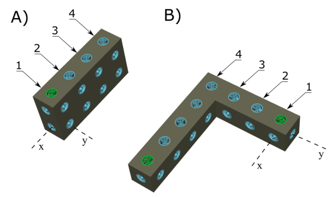

Figure 3: Visual Example of ‘Place_right_angle’ action: A) Initial state of robot before action is implemented. The top robot is the active robot and is the one performing the action. B) The final state of the robot after ‘Place_right_angle’ action has been implemented. The numbered annotations represent the discrete connection points on an robot and the axes signify the reference connection points on the robot

into an x-direction. This action can only be employed from the z-direction and into the xy plane.

Place_ceiling The action place_ceiling is employed to move a robot that is already vertical into a horizontal position on a higher z dimension, in effect constructing the next ‘layer’ of the lattice structure. The result of this action is -1 in the x, +4 in the z and no change in the y direction. The robot orientation also changes by 90 degrees into an x direction. The action can only be employed from the z direction and into the xy plane.

4 Planning Domain Representation

We aim to create a representation of the domain and plan output that is understandable and easy to adapt, whilst not oversimplifying the domain so as to reduce the usability of the system. As mentioned before, the domain representation is also intended to allow the creation of new actions in a modular fashion. For example, we want to make it easy to formulate the additional set of actions that might be required to bypass a wall via building bridge-like structures based on the primitives outlined in the previous section.

Unless specified otherwise, we use PDDL2.1 (Fox and Long 2011) (PDDL for brevity), the de facto standard planning domain specification language, to formulate planning domains. We also use the FD planner (Helmert and Domshlak 2009) because it is a competitive planner that allows multiple configurations. In the last International Planning Competition *IPC* (Torralba and Pommerening 2020), 28 out of 35 entries across the four classical tracks were based on FD’s code. FD grounds the lifted PDDL representation into FDR (Finite Domain Representation) as a pre-processing step. FDR, an extension of SAS (Jonsson and Bäckström 1998), is a grounded representation based on multi-valued state variables. When feasible, grounding improves planning performance but can become intractable for large problems.

The following description is a general explanation of the domain and the basic structure of how further actions can be created depending on the needs of the system in actual deployment.

Representation of the Multi-Robot System

As Figure 3 shows, each SR has four equally spaced connectors on each face that are integral to the structures that can be built. The robot must use these as connection points to create structures but also to create boundary conditions that the robot can work against when actuating. For example, for a ‘Traverse’ action to be deployed, a robot must be able to offer adequate connection points to the traversing robot. In this way, the connections can be considered the defining feature of the representation and become the coordinate space of the problem, which in turn sets the granularity of the domains. Using these connections as a coordinate space gives us the possibility to represent any robot as an object at a single location (x,y,z) with an orientation; the location corresponds to the first connector in the four, given a canonical convention, which is described below and represents our *first contribution* to 3D planning domains for MRSs.

The *canonical representation* is an important feature of the environment model as the robots are physically symmetrical in terms of their connection points. With a reference point as the first connection point (in Figure 3A denoted as connector 1), a robot that occupies Point(x,y,z) with an orientation of 0 degrees can be thought as the same as a robot, referenced to the same connection point, occupying a point at Point(x+L,y,z) (in Figure 3A denoted as connector four) with an orientation of 180 degrees (where L is the length of the robot projected in the X-axis). The symmetry encountered from a knowledge representation perspective is costly and unnecessary. The canonical representation ensures that these two functionally equivalent possibilities are always represented as one; a Point(x,y,z) with orientation of 0 degrees.

A crucial advantage of having a coordinate space based on the physical spacing of the connection points is that the representation is independent of changes to the physical dimensions of the robot. As long as the core attributes of equally spaced connections remain the same, the reasoning on this representation will also remain the same. The coordinate space also allows for a standardised framework that environmental data can be transformed to. In system deployment, the usable regions of the environment are discretised to the known ratio of the connection spaces.

The *second contribution* proposed for 3D planning in MRSs is the introduction of a *countable* robot representation. The core of this idea is to overcome the memory costs of requiring every action to be supported by a specific robot. Instead of defining each robot location individually ($at\_robot?\mathbf{r}?x?y?z\ldots$), where ‘?r’ is the actual robot we instead mark whether there is a robot on a specific set of coordinates ($at\_robot?x?y?z\ldots$). However, removing the robot ‘ids’ introduces a new problem, keeping track of how many robots are available at any given time to be inserted into the environments. We create a ‘countable’ predicate $is\_outside?c$ to keep track of how many robots are available at any point. As a new robot is introduced into the environment, this number is reduced by one. This representation reduces the memory usage in that obstruction logical checks are only required once per position instead of once per possible (position, robot) pair. This method is implemented by referring to the available robots as 'countable' objects in the domain representation.

These two contributions leave the critical definition of the domain to the following three object types:'coord', 'countable' and 'angle'. The type 'coord' is defined by the connector spacing mentioned above and effectively acts as the unit of a Cartesian coordinate system. The type 'countable' represents the SR available to the system and 'angle' is a set of constants that represent a robot's orientation. The problem is defined as a discrete uniform 3D grid whose size is defined by the maximum coordinate, e.g. a maximum coordinate of 10 corresponds to a 10x10x10 orthogonal grid.

We also model the environment as to have predefined points of entry: SRs can only enter the environment at a discrete number of locations. This assumption is coherent with the intended deployment conditions of the robotic system, where unstructured environments may only have a certain number of access points suitable to the robots.

Domain Syntax

A point (x, y, z) in space is represented by three objects of type 'coord'. Each dimension (x, y and z) is represented by a single 'coord', but 'coord' is axis agnostic so only a single range of objects is needed. Therefore, a problem definition in this domain has N objects of type 'coord', in an environment of (x, y, z) where $x = y = z = N$. An object of type 'angle' has 3 constants: 0ang, 90ang and 450ang, which correspond to the canonical representation detailed before.

The only predicates required to perform spatial reasoning within the environment are *plus_one*, *plus_three*, and *plus_four* e.g. *(plus_one ?x1 ?x2) (plus_one ?z2 ?z1)*, where point $(x2, y2, z2)$ has $x2 = x1 + 2$ and $z2 = z1 - 1$.

The following predicates are used in our domain to reason over the position and status of a robot:

- *at_robot(?x ?y ?z ?ang)*
- *dyn_at(?x ?y ?z)*
- *obstacle_at(?x ?y ?z)*
- *supported(?x ?y ?z)*
- *is_outside(?c)*
- *at_aperture (?x ?y ?x ?ang)*

The predicated *at_robot* has three 'coord' arguments and an angle. This allows a robot to be represented as a single point origin with a direction. This is a dynamic predicate which is updated after each action.

The predicate *obstacle_at* is used to identify an obstacle in the environment that is not a robot and takes three arguments of type 'coord' again.

The predicate *dyn_at* is used to denote the four locations that a robot will occupy when placed. It only represents a connector occupying a certain point (x, y, z). This predicate is used extensively for simplifying support and obstruction checks. In principle, the dyn_at predicate can be omitted; however, this results in significantly increasing the complexity of the support and obstruction checks as a function of the number of robots. The necessary checks only require to verify whether a point (x, y, z) is occupied by a robot or not. This can be obtained by the use of dyn_at in a single check, instead of many checks that would be needed if using at_robot. Our goal is to avoid unnecessary redundancies.

Utilising both predicates *at_robot* and dyn_at also allows for a richer goal language. It is possible to define the goal states by stating that: 1) an SR occupies a location with a defined orientation; 2) the MRS occupies several locations without specifying an orientation; and 3) both the previous cases. This way of specifying goals can lead to some interesting structures and increased flexibility for the human operator, who can answer different questions such as, for example, does the structure need to access a single point? Does the structure need to cover an area? Does the structure need to access a single point but exclude SRs at a set of locations?

The predicate *obstacle_at* is used to denote any environmental features that would constitute obstacles i.e. walls. It only represents an obstacle occupying a single point (x, y, z). This predicate is used extensively for simplifying support and obstruction checks.

The predicate *supported(?x ?y ?z)* serves a similar function to *dyn_at(?x ?y ?z)* but provides a different context to the state of a robot at a particular connector. The predicate considers structural soundness as a supporting point. Its use enables some actions to take place whilst excluding others that would create structural instabilities.

The predicate *is_outside(?c1)* denotes how many robots are available to be introduced to the environment and takes a single argument of object 'countable'. This value is updated when a robot is introduced to the environment and is supported by the predicate *robot_less(?c2 ?c1)*, which implements the logic of counting one down (like the previous *(plus_one ?x1 ?x2)* but with arguments of object 'countable', not 'coord').

The final predicate used is *at_aperture (?x ?y ?x ?ang)* that is used to show the single location in the environment where robots can be introduced into the environment. Although it represents a single coordinate, it must contain and 'angle' so that the robots introduced via the aperture can be correctly initialised in terms of their orientation.

Example Action To demonstrate the features of the domain mentioned before, the 'insert' action is represented below with an explanation.

```
(:action insert
    :parameters (?x1 ?y1 ?z1 ?p1 ?p2 ?p3 ?end_pos1 - coord
            ?c1 ?c2 - countable)
    :precondition (and
1     (is_outside ?c1)
2     (robot_less ?c1 ?c2)
3     (at_aperture ?x1 ?y1 ?z1 ang0)
4     (plus_one ?end_pos1 ?z1)
5     (plus_one ?x1 ?p1)
6     (plus_one ?p1 ?p2)
7     (plus_one ?p2 ?p3)
8     (not (dyn_at ?x1 ?y1 ?z1))
9     (not (dyn_at ?p1 ?y1 ?z1))
10    (not (dyn_at ?p2 ?y1 ?z1))
11    (not (dyn_at ?p3 ?y1 ?z1)) )
    :effect (and
12    (not (is_outside ?c1))
13    (is_outside ?c2)
14    (at_robot ?x1 ?y1 ?z1 ang0)
15    (dyn_at ?x1 ?y1 ?z1)
16    (dyn_at ?p1 ?y1 ?z1)
17    (dyn_at ?p2 ?y1 ?z1)
18    (dyn_at ?p3 ?y1 ?z1)
19    (supported ?x1 ?y1 ?z1)
20    (supported ?p1 ?y1 ?z1)
21   (supported ?p2 ?y1 ?z1)
22   (supported ?p3 ?y1 ?z1) ))
```

Lines 1 and 2 detail the logic surrounding the ‘countable’ method, with line 1 checking that there is a certain number of robots available to be inserted. Line 2 then enforces the finite number of robots available. The action will only apply if there is another robot available, i.e. *robot_less 7 6*), and will consequently reduce the number of robots available in the $is\_outside$ predicate. Once no robots are available outside (*is_outside 0*), then the action stops applying, i.e. it is not possible to keep inserting them. Line 3 checks that there is an aperture available to insert a robot, at a given location x, y, z. Lines 4,5,6 and 7 spatially reason to define the relevant positions to the insert action. As the aperture is defined to be in orientation ‘0ang’, the predicates are defined in terms of the x coordinate. Line 4 defines the z coordinate that is relevant, namely the space below the target for the robot to be inserted.

Lines 8,9,10 and 11 specifies that there should be no *dyn_at* predicates satisfied over the relevant x positions, with the y and z of the aperture. This means that there are no robot connectors occupying any of those spaces that might be blocking the execution of the proposed action. Note that this predicate removes the complexity of checking the single orientations of robot.

Lines 12 and 13 update the new number of robots available outside the environment. Line 14 places a robot at the aperture point in the corresponding orientation via the *at_robot* predicate. Lines 15,16,17 and 18 also update the corresponding *dyn_at* positions based on the *at_robot* predicate and lines 19,20,21 and 22 mirror these positions but define them as *supported* as the structural status of an inserted robot allows its four occupied positions to be *supported*.

The basic structure of the domain actions is reflected in the one presented above. The preconditions define the relevant positions for the proposed action and check that no obstruction will occur when the action is performed as well as that the required supports are available. The effects remove the old occupancy information and update it with the new positions by using the *at_robot*, *dyn_at* and *supported* predicates.

5 Experiments

We compare four domain representations in the following experiments. The purpose is to evaluate the efficacy of our representation with respect to the contributions detailed in Section 4. The domains differ in the adoption of these contributions. The notation described next is relevant for Figures 4 & 5. The Canonical domain is the domain representation that contains both of the contributions - the countable robot representation and the canonical orientations, referred to as **CaCo**. The Non-Canonical domain is the domain representation that contains only the countable robot representation, referred to as **NCaCo**. The Agent-Canonical domain is the domain representation that contains only the canonical orientation, referred to as **CaNCo**. The Agent-Non-Canonical domain is the domain representation that does not contain any of the contributions of the canonical representation, referred to as **NCaNCo**.

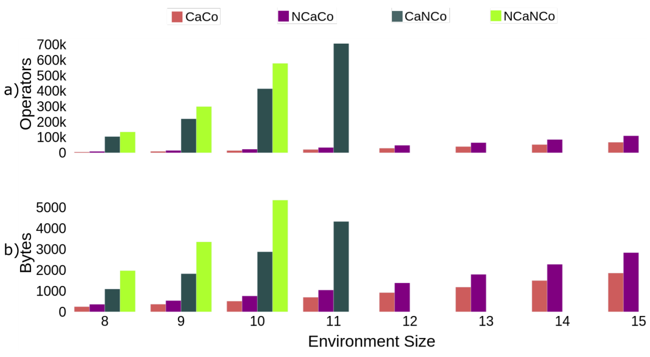

Figure 4: a) Operator and b) Bytes size by environment size and domain variant. The absence of plots show that the planner failed at all instances.

Setup

We design the experiments to evaluate the effectiveness of the overall approach with respect to scalable and accurate planning in 3D environments. All experiments were carried out on a cluster of Intel Xeon E5-2640 running at 2.60GHz. The memory limit by process was 12 GBs and the time limit was 1,800 seconds. The primary metric by which the representations are evaluated is coverage. All experiments follow the same format: a randomised goal set in a fixed-size environment with a given number of SRs available to the planner. Conceptually, the goal is to reach a specific target coordinate where the service robot needs to perform some tasks. This is specified as reaching any one of the perpendicularly adjacent coordinates that would allow the robot to work on the target coordinate. Since this leaves a maximum of six possible alternative goals, depending on goal location and obstacles, the set is treated as an ‘or’ statement. This method is used to facilitate the description of goals that involve reaching a specific coordinate in the environment so as to perform tasks on it.

We present both optimal and sub-optimal search experiments. For optimal search, we used the FD’s implementation of A* (Helmert 2006) and the heuristic h_{max} (Bonet and Geffner 2001), which is admissible for domains with axioms and conditional effects. For sub-optimal search, we use LAMA. We have tried to employ the winner of the last IPC(Torralba and Pommerening 2020) satisfying competition, StoneSoup2018 (Seipp and Röger 2018), but LAMA has a better coverage for this domain.

The complexity of the Domains

Figure 4 shows both the memory usage (bytes) and number of grounded operators for each domain for grid sizes (GS) varying from $GS = 8$ $(8 \times 8 \times 8)$ to $GS = 15$. Note that for bigger GS values, agent-based representations are missing because grounding runs out of memory. The canonical domain uses the smallest amount of memory in all cases. We set the number of SRs to be equal to GS for three reasons. Firstly, it simplifies the complexity analysis because then there is only one variable (GS). Secondly, the bigger the grid the more SRs needed on average to reach a randomized goal. Finally, agent-based representations grounding performance decreases as the number of SRs increases, hence in order to make an unbiased comparison we only increase the number of SRs when the grid-size requires it.

The equations below show the state's bit usage as a function of GS for each representation.

$$M_{CaCo} = 2*GS^3 + 3*(GS^2*(GS-3)) + GS + 1 \quad (1a)$$

$$M_{NCaCo} = 2*GS^3 + \mathbf{6}*(GS^2*(GS-3)) + GS + 1 \quad (1b)$$

$$M_{CaNCo} = 2*GS^3 + 3*(GS^{\mathbf{3}}*(GS-3)) + GS + 1 \quad (1c)$$

$$M_{NCaNCo} = 2*GS^3 + \mathbf{6}*(GS^{\mathbf{3}}*(GS-3)) + GS + 1 \quad (1d)$$

In Eq. (1a), relating to the Canonical representation, the term $2 * GS^3$ represents the memory usage of both dyn_at and $supported$ predicates; in particular, there is a fact per point (x, y, z) specifying the predicate's truth value. The term $3*(GS^2*(GS-3))$ corresponds to the three at_robot predicates (each predicate applies to all points that are at least three units away from the grid's border on its corresponding orientation; otherwise, some of the robots would be outwith the grid). The term $+GS$ corresponds to the $is\_outside$ binary variables. There are as many variables as the number of robots in the problem. The final term $+1$ corresponds to the axiom variable used to represent whether goal conditions are met.

Eq. (1b) relates to the Non-Canonical representation. In this case, the number of variables used to represent the at_robot variable is doubled due to the dual representation of robots' orientations. For the Agent-Canonical representation (Eq. (1c)), the number of grounded at_robot is increased by a factor of GS because the at_robot now specifies the robot's identity. Finally, for the Agent-Non-Canonical representation (Eq. 1d), the number of at_robot variables increases by a factor of $2 * GS$ due to the doubling of at_robot variables in the Non-Canonical representation and the factor increases of GS due to the agent-based representation.

There are too many lifted operators (41 in the Canonical domain) to make a similar, detailed symbolic-formula-analysis practical. However, Figure 4a) shows that the Canonical domain uses the least amount of grounded operators. At $GS = 10$, the biggest grid size at which all domains are successfully grounded, the Non-Canonical, Agent-Canonical and Agent-Non-Canonical domains have more operators than the Canonical domain by a factor of 1.62, 28.73 and 40.03, respectively.

Results

Figure 5a) shows that the consistently best performing domain is the Canonical representation. As the environment size grows, it can be seen that all domains perform worse, but the Canonical variant still maintains the highest coverage. This is true for both the optimal search with h_{max} and the sub-optimal search with LAMA.

As the primary goal of this paper is to show both an accurate and a scalable domain representation for planning in 3D, it is important to look at how the four variants perform in terms of scalability. Figure 5b) shows how many problems are unsolved because the planner fails while grounding the representation, and Figure 5c) shows when the planner fails while searching for a solution.

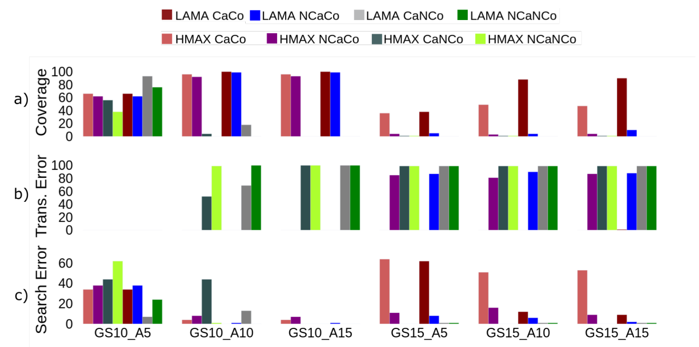

Figure 5: a) Coverage, b) translation error and c) search error by domain variant, environment size, number of agents, and search method (A* with h_{max} or LAMA). The environment size and number of agents are denoted by GS and A respectively. The absence of plots show that the planner failed at all instances.

As previously said, FD has a pre-processing phase where PDDL is translated from lifted PDDL to grounded FDR; grounded planners need to ground the domain before they can search for a solution. Figure 5b) shows that when the canonical problems fails to solve, it is because the search phase fails to find a solution. However, as GS is increased, the other variants fail due to grounding. This shows that the Canonical version is the more scalable of the options in practice, since a solution is searched for in all cases.

The other three variants demonstrate the unsuitability of those domain representations for planning. The Non-canonical variant (NCaCo) is the second best performing, with reasonable coverage achieved for an environment up to size 10. In fact, the coverage between the Canonical and Non-canonical for GS10_A10 and GS10_A15 have very little difference. It is however apparent that the canonical orientation feature is critical for scalability of the system as the coverage worsens considerably when moving to the larger environment size. These results show that the complexity savings in introducing a canonical orientation in the domain are almost as significant in grounding scalability as in memory usage per search state. These observations are further confirmed by their performance using LAMA where the pattern is further observed.

The agent domains, in both forms CaNCo and NCaNCo, perform poorly, as expected (see Eq. 1c and 1d). Achieving poor coverage as the number of agents increases, as seen in Figure 5a), and showing errors in the search phase (Figure 5c)) demonstrates that utilising the 'non-countable' object representation for 3D robotic problems is not a viable option. This is a significant point to make as, although it may be an intuitive representation, it will significantly hamper planning efforts to operate in the MRS Task/Path planning field.

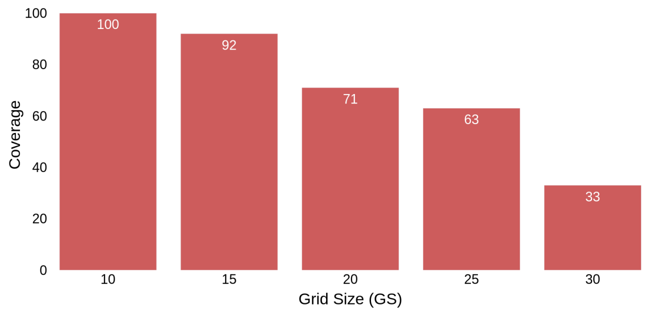

Figure 6: Coverage of Canonical domain variant as environment size increases with fixed number of agents at 100. Experiments performed using direct FDR file generator

We take full advantage of the homogeneity of this system, i.e. if every robot had a different functionality our countable approach would not be applicable. In some cases, there will be different robot 'categories', for such cases our countable approach would be separately applicable to each of these categories.

As the Canonical variant can be seen to be the best performing variant, we generate the FDR representations directly, via a python script instead of PDDL, to bypass the translation phase requirements. To further assess our claim that this domain representation provides a solution for accurate and scalable planning in 3D, more experiments have been conducted to push the domain to scales reasonable for the MRS in real-world deployment. To avoid a similar coverage issue as seen in GS10_A5, domains were given 100 robots regardless of the environment size. The goal sets were constructed and randomised as in the previous set of experiments. The full results can be seen in Figure 6.

The coverage shown in Figure 6 declines in a linear fashion as the environment size increases. A step change in the coverage can be seen for an environment of size 30, suggesting that our method is approaching its limits at that size. This is, however, a relatively large grid size for the practical application of a single robot that occupies four positions. The largest environment (GS=30) has 27,000 points while the largest PDDL environment (GS=15) has 3,375 points. At $GS = 30$, a maximum of 225 robots could occupy the environment.

Complex Environments

The experiments above have been performed in an empty environment without natural features that could act as obstacles to the MRS. Figure 7 shows a demonstration of how the proposed AI system can make use of the predicate *obstacle_at(?x ?y ?z)* to represent obstacles in the environment. In this example, there is a continuous wall that runs across the entire environment (cross-sectioned for the purposes of demonstrating the resulting structure). It cannot be moved around and so must be climbed over. The resulting structure resembles a bridge and successfully overcomes the obstacle.

This plan is produced by adding the necessary actions to allow for bridge-like structures to be formed and illustrates the aforementioned modularity of the domain representation. It also allows complex actions to be utilised; as it can be seen in Figure 7, the planner has produced a standalone extension of a robot to allow for connection points to be where they must be to satisfy connection requirements for necessary actions. Figure 8 demonstrates a useful cubic structure. This requires a complex plan which is found by combining LAMA and the CaCo representation.

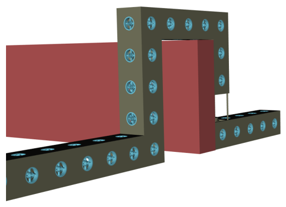

Figure 7: Example structure to overcome a sample obstacle, like a continuous wall, leading to a 'bridge' like structure. **Video** https://youtu.be/419tBz6bVxw

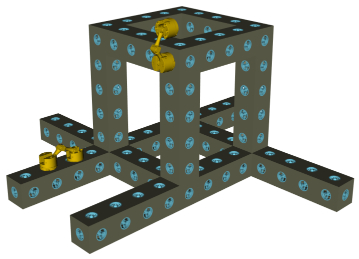

Figure 8: Cube, a complex to build structure which can enable service robots access to previously unreachable areas. **Video** https://youtu.be/rnW0LnPUgMo

6 Conclusion

In this paper, we demonstrate two contributions (canonical, countable) to the problem of 3D planning for an MRS. In the key issue of producing an accurate and scalable domain representation, we have shown comprehensively that a countable agent representation and canonical orientations significantly reduce the state size of a given problem and outperforms the other domain representations that do not incorporate these features. We also show that this domain representation is scalable to practical environment sizes, when the translation phase of PDDL is removed by directly writing FDR representations. Scalability could be furthermore increased by using domain-specific heuristics. This is left for future work. We show empirically that the introduction of the two domain contributions described is critical for the effective application of 3D planning of MRS.

7 Acknowledgments

InnovateUK Connect-R (TS/S017623/1). EPSRC and the CDT in Robotics and Autonomous Systems at Heriot-Watt and The University of Edinburgh (EP/L016834/1).

References

Alkhazraji, Y.; Katz, M.; Matmüller, R.; Pommerening, F.; Shleyfman, A.; and Wehrle, M. 2014. Metis: Arming fast downward with pruning and incremental computation. *International Planning Competition (IPC)* 88–92.

Asl, A.; and Davis, E. 2014. A qualitative calculus for three-dimensional rotations. *Spatial Cognition & Computation* 14(1): 18–57.

Balakirsky, S.; Kootbally, Z.; Schlenoff, C.; Kramer, T.; and Gupta, S. 2012. An industrial robotic knowledge representation for kit building applications. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1365–1370. IEEE.

Belouaer, L.; Bouzid, M.; and Mouaddib, A.-I. 2011. Spatial knowledge in planning language.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence. 2001 Jun; 129 (1-2): 5-33* .

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

Churchill, D.; and Buro, M. 2011. Build order optimization in starcraft. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Dornhege, C.; and Hertle, A. 2013. Integrated symbolic planning in the tidyup-robot project. In *2013 AAAI Spring Symposium Series*.

Estivill-Castro, V.; and Ferrer-Mestres, J. 2013. Path-finding in dynamic environments with PDDL-planners. In *2013 16th International Conference on Advanced Robotics (ICAR)*, 1–7. IEEE.

Fox, M.; and Long, D. 2011. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *CoRR* abs/1106.4561. URL http://arxiv.org/abs/1106.4561.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Nineteenth International Conference on Automated Planning and Scheduling*.

Jenett, B.; Abdel-Rahman, A.; Cheung, K.; and Gershenfeld, N. 2019. Material–Robot System for Assembly of Discrete Cellular Structures. *IEEE Robotics and Automation Letters* 4(4): 4019–4026.

Jonsson, P.; and Bäckström, C. 1998. State-Variable Planning Under Structural Restrictions: Algorithms and Complexity. *Artif. Intell.* 100(1-2): 125–176. doi:10.1016/S0004-3702(98)00003-4. URL https://doi.org/10.1016/S0004-3702(98)00003-4.

Koehler, J. 1998. The 1st International Planning Competition:Gripper Domain. https://ipc98.icaps-conference.org/. Online, accessed Dec-2020.

Kotay, K.; Rus, D.; Vona, M.; and McGray, C. 1998. The self-reconfiguring robotic molecule. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, volume 1, 424–431. IEEE.

Leidner, D.; and Borst, C. 2013. Hybrid reasoning for mobile manipulation based on object knowledge. In *Workshop on AI-based robotics at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Munoz, P.; R-Moreno, M. D.; and Castano, B. 2010. Integrating a PDDL-based planner and a PLEXIL-executor into the ptinto robot. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 72–81. Springer.

Quintero, E.; Alcázar, V.; Borrajo, D.; Fdez-Olivares, J.; Fernández, F.; García-Olaya, Á.; Guzmán, C.; Onaindía, E.; and Prior, D. 2011. Autonomous mobile robot control and learning with the pelea architecture. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Riddle, P. J.; Barley, M. W.; Franco, S.; and Douglas, J. 2015. Automated Transformation of PDDL Representations. In Lelis, L.; and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel*, 214–215. AAAI Press. ISBN 978-1-57735-732-2. URL http://www.aaai.org/ocs/index.php/SOCS/SOCS15/paper/view/11166.

Romanishin, J. W.; Gilpin, K.; Claici, S.; and Rus, D. 2015. 3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 1925–1932. IEEE.

Seipp, J.; and Röger, G. 2018. Fast downward stone soup 2018. *IPC2018–Classical Tracks* 72–74.

Stewart, R. L.; and Russell, R. A. 2006. A distributed feedback mechanism to regulate wall construction by a robotic swarm. *Adaptive Behavior* 14(1): 21–51.

Takahashi, T.; Lanighan, M. W.; and Grupen, R. A. 2017. Hybrid task planning grounded in belief: Constructing physical copies of simple structures. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*.

Torralba, A.; and Pommerening, F. 2020. Repository of ICAPS IPC-2018. https://ipc2018-classical.bitbucket.io/#. Online, accessed Dec-2020.

Toussaint, M. 2015. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Wolter, D.; and Kirsch, A. 2015. Leveraging qualitative reasoning to learning manipulation tasks. *Robotics* 4(3): 253–283.