UNIVERSITY OF BIRMINGHAM

Towards An Holistic Approach for Highly Flexible Robotic Assembly Systems

Mark Robson

October 2023

Contents

Summary/Abstract

This thesis presents a general framework for the control of a robotic agent tasked with solving problems in the domain of industrial assembly. A set of motivating example problems are described which shape the development of the framework. The framework fuses the concepts of a closely coupled, multi-layer reasoning architecture with abstract assembly sequence planning. This framework provides a reusable and flexible structure for robotic manipulation via a modular location graph, and a grasp planning mechanism that considers object and task constraints.

One example assembly task inspired by the building of light gauge steel frame panels for the construction sector is used to develop a benchmark object set and challenge assemblies. These objects are published as the open source Robotic Assembly Manipulation and Planning (RAMP) benchmark [1].

The abstract task planning layer finds valid sequences of part addition to complete an assembly task given a task description specified using Action Language $\mathcal{AL}_d$ [2] sequentially feeding these steps as subgoals to a closely coupled robotic reasoning framework based on the REBA architecture [3]. Experiments demonstrate application to long time horizon assembly problems requiring the addition of many parts with over 200 low level robot actions. The abstract task planning layer reduced coarse-resolution planning time by 93.5% compared to a baseline which must simultaneously consider assembly sequencing and robot actions.

The modular location graph structure links positional information between the task space and logical domain. We utilise this structure to parameterise the assembly actions of the RAMP benchmark tasks. Additionally, a pruning heuristic is proposed to speed up searching in this location graph when the robot is planning assembly motions.

A grasp planning approach is detailed utilising a weighted grasp scoring model considering a combination of measurement uncertainty and variation in the extracted surface, the contact angle of gripper fingers to the surface, as well as task constraints. A three-level representation for objects, compatible with our framework, includes object class membership, point cloud data representing the objects surface, and semantic keypoints linked to the object parts. A learned model is used to encode task-specific knowledge from a small number of exemplars of objects, tasks, and relevant grasps; preserving the relationship between keypoints and grasp points for specific tasks despite changes in factors such as the scale and orientation of objects. The learned models are queried at run time to guide the generation of grasps that balance task and stability constraints. Through experimental evaluation on a Franka robot manipulator with a parallel gripper, it is demonstrated that this method is able to generate grasps on previously unseen objects achieving the desired task-specific trade off whilst maintaining a high degree of grasp stability.

Statement

This thesis is submitted for the award of Master of Philosophy in Computer Science. The content of this work expands upon two pieces of work submitted for review to academic publications. Firstly, **A Keypoint-based Object Representation for Generating Task-specific Grasps** presented as a conference paper at the IEEE 18th International Conference on Automation Science and Engineering (IEEE CASE2022) [4] which is entirely my own work with supervision from Mohan Sridharan. Secondly, **RAMP: A Benchmark for Evaluating Robotic Assembly Manipulation and Planning** [1], submitted to IEEE Robotics and Automation Letters, which was jointly authored by Jack Collins, Jun Yamada and myself with supervision from Mohan Sridharan, Karol Janik, and Ingmar Posner. My contributions to this work included all aspects of the planning system and the framework structure for the baseline system, the concept of using the Light Gauge Steel Frame construction beams as the inspiration of the benchmark components, xml assembly description formulation, and software for parsing the xml data to determine object poses in space required to assemble the beams with the robot. Design of the components was completed in collaboration between myself and Jack Collins. Jack Collins created an accompanying simulation environment and developed the benchmark scoring and submission mechanism, Jun Yamada implemented the low level control for actions for the robot and conducted the trails.

Acknowledgements

This work was completed part-time alongside full-time professional employment. The author would like to thank The Manufacturing Technology Centre for supporting this work through sponsorship and the provision of time to allocate towards research.

I would also like to thank my supervisor Mohan Sridharan who has provided much support and guidance as well as my collaborators Jack Collins, Jun Yamada and Ingmar Posner of the Oxford Robotics Institute and Karol Janik of The Manufacturing Technology Centre.

1 Introduction

Consider an industrial robot which must select, pick-up, and assemble a set of parts to build a product in the fastest time possible and without mistakes. Industrial robots must achieve over 99.9% error free performance and minimise cycle time in order to maximise productivity. Now, imagine that the assembly being built is a set of steel construction beams which together build a roof or wall panel structure which will later be craned into a building. Depending on elements of the building's design such as the room layout, window and door positions, or locations of plumbing or electrical systems; each panel may be subtly or drastically different in its configuration, but the types of beams and joints used will be similar. We will return to this task, alongside other motivating example tasks, to illustrate how the work completed in this thesis may be applied.

In order to enforce high performance in industrial settings engineers have typically "designed out" variation through the development of sensors, fixtures, customised grippers, feeding systems, etc. Presenting robotic assembly systems, such as those found in automotive body in white lines, with a consistent and repeatable world in which they perform assembly tasks though the repetition of the same set of pre-programmed actions. This approach, called "hard automation", is expensive and as such requires significant volume or value of products to be produced in order to justify the set up costs.

Increasingly, to make further productivity gains and automate in lower volume, or lower product value scenarios, more flexible robots capable of handling several tasks whilst requiring lower capital investment are becoming crucial. Collaborative robots and improvements to the ease of re-programming have already made an important contribution to increasing robot density in many industrial environments, largely through automation of simple pick and place applications such as palletising or machine tending. However, there are still many opportunities for robots which can work more interactively with people, and which can perform more complex tasks.

There are several key concepts which are important for robots to tackle these more challenging and varied assembly tasks.

- Perception; robots must be able to understand their surroundings, and find objects of interest within their operating space.
- Planning; robots must generate reasonable and reliable plans of action i.e. they must be able to determine for themselves the sequence of actions to take in order to achieve a goal. They must also be able to detect scenarios where things have not gone to plan, for example when an action has not achieved the desired outcome, and re-plan to get back on track.
- Object manipulation; robots must be able to manipulate various objects, as humans do, to pick objects up, move them safely through the work space, and control contact rich interactions between the object and environment in order to assemble objects together.

1.1 Problem Statement

To achieve complex goals, robots need task planning algorithms to break down problems into sequences of executable skills. Long time horizon planning and robot skill control are a dual problem as planning relies on the predictable outcome of skill execution, whilst individual skill execution relies on good preparation achieved though previous actions. For example to successfully insert a beam into our assembly example the robot must previously have grasped the beam in a suitable place to complete the necessary insertion action without collision between the robot gripper and other beams already in the assembly.

One of the challenges of robotic assembly is to coordinate the perception, task planning and motion planning components that are essential for achieving the desired outcome. Task planning involves determining the sequence of actions that need to be performed to assemble the parts into the desired structure. For assembly of multi-part assemblies many actions may be required presenting a difficult challenge as these systems must consider large possible states over many time steps. Task planning in such domains can quickly become intractable due to the exponential increase in the state space over subsequent time steps. Imperfect models, and uncertainty in perception and execution of robot actions also present challenges in successful completion of long-horizon tasks.

Robot in-hand manipulation remains difficult as robot end effectors (or hands) lack the dexterity or sensitivity of their anthropomorphic equivalents. Often it is preferable for a robot to grasp objects in such a way that in-hand manipulation of the object is not required or so that re-orienting and re-grasping the object is made more feasible. Robotic grasping is the action of attaching an object to the robot's end effector though the forming of contacts between end effector and the target object. Robotic grasping presents a challenge of planning and executing suitable robot grasps based on object-task relationships in a way which generalises to many object types that a robot might be expected to interact with in an industrial environment.

In this thesis we address the dual problem of robot skill control and long time horizon planning. Specifically we look at applying a general methodology to long time horizon robot manipulation problems and performing robotic grasping of objects given some knowledge, or plan, of future object-task behaviours.

1.2 Aims of the Study

This study aims to present three main contributions:

1. A general reasoning and control framework widely applicable to robotic assembly problems
2. A novel method for incorporating task data in grasp planning
3. A scalable benchmark which tests long horizon planning, reasoning and control for robot assembly

1.3 Definition of Manipulation Tasks

Any task where an object is physically moved by an external system can be described as a **Manipulation Task**. Object manipulation covers a wide range of interactions including re-positioning an object in a workspace, as well as more complex interactions with objects such as reconfiguring moving or deformable object parts by applying forces to the object. We define robotic manipulation tasks as any task where a robot must control the interaction between an object and the environment or other objects in the environment. Manipulation tasks are object-centric, in that they can be described by the desired movement of the object. The following are all examples of robotic manipulation tasks:

- Acquiring and lifting an object from a table
- Using a tool e.g., hammering a nail or driving a screw
- Pushing or sliding a block along a surface to a target destination
- Throwing or tossing objects at a target
- Insertion tasks e.g., placing a peg into a hole
- Squeezing a spray bottle trigger
- Twisting one row of a Rubik cube
- Moving a held object through space

We consider **Assembly Tasks** as a subset of robotic manipulation tasks where the robot already has an object in its grasp and this object must be brought into a contact interaction with one or more other objects in order to fit them together into a desirable configuration. The intention of an assembly task is to achieved a desired outcome state of the object set. Assembly tasks might include:

- Single or multiple peg in hole insertion
- Screwing a lid onto a container
- Stacking objects into a pile

Authors writing on mechanical design for assembly [5] [6] present detailed analysis of assembly or "fitting" task difficulty with features of the task interaction governing a scoring metric for how difficult to assemble a single part is, the intention of these works was to guide engineering design towards easier to assemble structures, thus facilitating higher assembly productivity. Features affecting difficulty include; straight-line v.s. non-straight-line assembly, top-down v.s. non-vertical insertion direction, ease of alignment, resistance to insertion, and single v.s. multi-site insertion.

Grasping is a key skill for enabling robot manipulation interactions with objects. **Robotic Grasping** can be defined as the act of forming a persisting robot-object relationship using an end effector, with the intent of lifting/transporting the object though free space. Grasping using a generic gripper opens up the possibility of interacting with a much wider range of objects and **Dexterous Grasping** specifically refers to grasping through the use contact between the object and multiple fingers of the end effector as opposed to the use of suction or other grasping methods.

2 Related Work

Symbolic knowledge representations are expressive for reasoning about declarative knowledge including describing relations and constraints of task planning. Since Shakey, the robot used to test the AI planning language STRIPS [7], approaches for symbolic task planning have intersected robotics and AI research. Declarative languages for formalising the task planning domain include Answer Set Prolog (ASP) [8] and Planning Domain Definition Language (PDDL) [9].

Mechanisms for reasoning often incorporate hierarchical levels of abstraction to make planning more efficient [10]. Methods include task decomposition such as Hierarchical Task Network (HTN) planning [11] or more expressive formulations such as using multi-level knowledge representations [3]. Hierarchical abstraction is a useful structure in planning for assembly tasks where there must be some inherent order to the assembly process. For example, we can reason about the beam assembly definition to generate a high-level plan encoding a viable assembly sequence, and reason at finer granularity about geometric locations to grasp and manipulate the beams.

Traditionally, a descriptive model of the domain is specified by the designer. More recently researchers have looked at learning these models. Learning-based methods can be used to generate planning models, to identify heuristics for guiding search, and to acquire knowledge about unknown action effects or preconditions [12] [13]. However, purely learning-based methods remain data-inefficient in long-horizon problems.

There are many general cognitive architectures with varying approaches to separating high-level task planning, low-level motion planning and observations including ACT-R [14], SOAR [15], ICARUS [16], DIARC [17] and T-REX [18]. For example, the T-REX architecture implements a sense-deliberate-act cycle synchronising several concurrent control loops as Teleo-Reactive programs over unit time steps. Implementations of the T-REX architecture using variants of Temporal PDDL planners have been demonstrated for control of unmanned underwater vehicle, orbital satellite robot arm, and exploration rover domains [18, 19].

One popular approach is to use the SkiROS2 architecture due to its integration with the latest ROS2 middleware. SkiROS2 uses a modular approach with a separate PDDL based planning module to generate an ordered sequence of actions with skill execution controlled by a behaviour tree [20]. Behaviour trees can be complex to implement but have been shown to model the expressiveness of Finite State Machines, Hierarchical Task Networks, Subsumption Architectures, Teleo-Reactive programs, and Decision Trees [21].

The REBA architecture, which we choose to build upon in this thesis, provides a method for reasoning at two hierarchical levels. REBA uses the SPARC ASP formulation [22] for knowledge representation, non-monotonic reasoning, and diagnostics. Formal coupling of the hierarchical layers of the reasoning model allows for more detailed constraints to be considered only when necessary, by automatically selecting only the relevant objects from the domain when *zooming* into the fine-resolution model. REBA also provides a method for implementing observation actions to update the agent's belief state in the reasoning model. This can provide continuously reactive control, for example by implementing robot action controllers using automatically generated partially observable Markov decision processes (POMDP)'s [3].

The key difference between prior work and REBA is in how REBA formulates and addresses the problem, i.e., in reasoning about the domain at two (or more) formally coupled abstractions. This coupling helps make reasoning and learning more reliable and efficient.

Recently, generative Large Language Models (LLMs) have demonstrated impressive common sense reasoning capability and as such interest in applying this technology to robotics has been increasing. Advantages for general robotic agents include the ability of such models to infer concepts from large data repositories such as relations and affordances of objects in a scene [23], and to classify objects in order to take more general, human language instructions [24]. Currently, LLM's have known shortcomings for example suffering from confabulations where they produce inaccurate responses based on inaccuracies in what LLM's seem to "think" they "know". As such LLM's have only been demonstrated on limited complexity tasks in terms of long horizon interactions.

Perception, grasping, and assembly actions are critical skills to enable the robotic agent to execute and monitor planned tasks in a physical domain. The robot must be able to perceive objects in its environment in order to identify items to manipulate and to collect information to be used in subsequent steps such as a point-cloud of the object surface on which to plan grasps. Perception may also allow the robot to assess if its actions have completed successfully or not, in combination with a suitable non-monotonic reasoning mechanism this capability allows the robot to recover from failure [3].

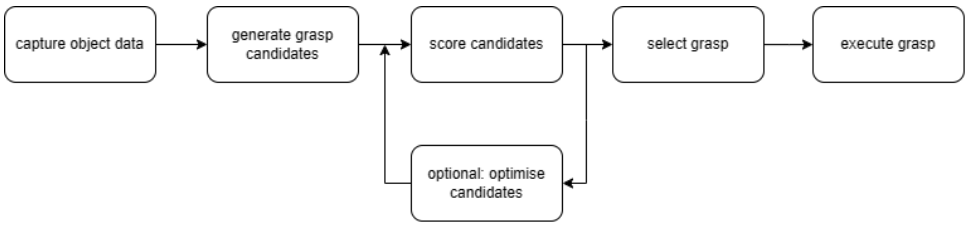


Figure 1: A generalised pipeline for robotic grasping.

Assembly tasks are a subset of manipulation skills which can be parameterised by the desired object motion between states. Some tasks may be controlled in an open loop e.g., motion through free space whilst others might require closed loop control. Perception via robot proprioception, addition of force-torque sensing at the end effector, tactile sensing and external sensors such as cameras are all commonly used in combination with mathematical models to control robot motion. Many works rely upon pose detection to assess object state at each time step, optical markers such as Aruco and April Tags are commonly used for this purpose. 6DOF pose estimation from RGBD images or pointcloud data, and the construction of a class level semantic keypoint skeleton [25, 26] have also been used to model and control the desired object motion. Assembly skills might be manually parameterised, learned from human examples via teleoperation, or learned in simulation [27]. Some assembly skills are generally applicable such as peg-in-hole insertion, and may be parameterised to be reused for multiple scenarios. However, individual assembly skills might also be unique to the product being constructed, as such it is important that our framework be able to flexibly support a wide variety of assembly skills.

Grasping is a key skill for enabling the robot to interact with objects in the environment. It is difficult to compare grasping methods from literature because grasp success measures are often task and domain dependent. It is clear however that state of the art methods have transitioned from analytic methods towards data-driven approaches [28]. A generalised pipeline for planning robotic grasps is shown in Figure1. Both analytic and data driven methods generate feasible grasp candidates and use a scoring system based on mathematical models, heuristics, learned models or a combination thereof to select the best grasp to execute.

Methods using deep neural networks with RGBD images as input have reported high success rates for grasping novel objects with a two finger parallel gripper [29–32]. However, many of these approaches are limited to a single approach direction, typically from above. This is limiting for manipulation tasks where the 6DOF pose of the gripper w.r.t the object held may have a significant impact on the robots ability to manipulate the object.

Compared to parallel grippers, finding optimal grasp candidates for robot hands with multiple degrees of freedom pose a high-dimensional search problem. Dimensionality reduction methods have been explored to identify good grasps which tend to cluster in this search space [33, 34]. Generative methods have also been developed to sample the space of grasp candidates, with some methods initialising grasp candidates based on previously successful grasps before optimising for hand configuration and finger placement [35–38].

The choice of a good grasp is essential for the success of automated robot grasping and manipulation [39, 40]. Considering task requirements may result in a less optimal grasp in terms of stability but it may increase the ability to manipulate the object as required by the task [41, 42]. Knowledge of manipulation trajectory can also be used to impose checks on kinematic feasibility of grasps at the start and end poses thereby reducing the grasp space considerably [43].

Methods developed to specify task-specific constraints and regions suitable for grasping an object learn these regions from simulation trials [44–46], from large numbers of labelled images [47], or as an abstract function that defines task-specific approach vectors for object classes [48, 49].

Whilst symbolic representations are powerful for logical reasoning, continuous representations such as joint angles and torques are necessary for direct control of the robot at an actuation level. Task and Motion Planning (TAMP) attempts to integrate high-level task planning with low-level motion planning to generate feasible action sequences for the completion of long-horizon tasks. We direct interested readers to a recent review of TAMP approaches [50]. We choose to separate task and motion planning intentionally to allow for manipulation skills of any kind to be operable with our reasoning system approach. This is an important distinction for making our framework applicable to many robot agents, and many assembly problems.

There are relatively few benchmarks for long-horizon assembly tasks with many researchers developing their

own tasks in order to test developed approaches. Multiple object collections exist for grasp planning such as the YCB dataset [51] and object sets from the Amazon Picking Challenges [52] but these objects may be difficult to source in some regions and typically the objects do not lend themselves to assembly or manipulation tasks requiring many time steps. IKEA furniture has been used to provide a comparatively accessible and repeatable object set requiring longer horizon manipulation sequences to progress assemblies [53]. For such furniture assembly tasks it is hard to define distinct levels of complexity or difficulty and there is no guarantee that commercial entities will continue to sell the associated furniture indefinitely. Other works have looked at timber framing for construction which requires consideration of both robot skills [54] and assembly sequencing to maintain structural stability [55]. Several researchers have used these object sets to investigate robotic skill learning but fewer have used them to consider the robots reasoning framework more holistically.

Whilst researchers may wish to focus on specific areas of interest, robotic agents capable of completing assembly problems require complete system solutions; combining perception, planning and control. The current benchmarks fail to facilitate comparison of different approaches to the holistic challenge of robotic assembly over long-horizon tasks. As such, a new benchmark is necessary which represents industrially relevant assembly tasks, is scaled appropriately for use with common academic robotics hardware, and can be easily recreate-able at low cost.

3 Benchmark Tasks

In order to shape our framework three example tasks were considered which provide challenging scenarios for a robotic agent. These tasks are based on industrial robotic assembly tasks. The scale and object sets of the tasks have been simplified in order to make the tasks more applicable to type of equipment and setups available in many academic labs, namely standalone collaborative manipulator robots with a tabletop work space. The three tasks provide a range of complexities with which a general robotic assembly agent must be able to contend. We will return to the tasks throughout this work to consider how the research illuminates possible routes to a general solution.

1. Shape Puzzle - Bin picking introduces a high degree of uncertainty to the order of objects which can be retrieved by the robot which it must sort through to assemble parts in the desired sequence
2. Tool Tidy - A collaborative robot helper for handling workshop tools putting them away in a shadow board, rearranging them on a table surface, or presenting them to a person.
3. Beam Assembly - A robot assembly challenge using a tabletop scale representation of light gauge steel (LGS) framing, used commercially to make construction panels. Individual beams can have a multitude of joints which place constraints on assembly order and viable robot holding configurations.

The remainder of this section describes the individual tasks in more detail.

3.1 Shape Puzzle

Imagine a robot which has on one side a children's shape puzzle, illustrated in Figure 2, on the other side it has a bin full of shapes. The robot's task is to complete the puzzle by placing shapes, into the correct pockets, in a specified order. As such, the robot must search through the shapes from the bin to find the right ones to complete the puzzle. The addition of an ordering constraint reflects the sequential nature of many industrial assembly tasks. As the input shapes are lacking order in the bin this encourages the exploration of time/action efficiency trade offs and planning with uncertainty, despite the simplicity of the object handling element.

This pick and place task is actually surprisingly common in industry as many goods are produced and shipped in bulk but presented to consumers in packaging that holds a set of parts, for example Easter eggs, bathroom kits, and cutlery sets to name but a few. Outside of consumer products in the automotive and aerospace industries, kits of parts are often made up manually from bulk so that the assembly workers are presented with exactly what they need in pre-defined locations in order to maximise efficiency on the production line.

In order to add some variety we add some shapes with handles, and others which are flat, this introduces a requirement for the robot to change grasping methodologies. In order for the robot to sort through the shapes we allow the robot to place objects on the table surface at a location between the bin and the puzzle.

3.2 Tool Tidy

In a workshop environment, a collaborative robot helper should be able to retrieve and pass tools to a person on demand, the robot might also rearrange the tools on a table surface to make finding the right tool easier for the human worker to find, for example placing spanners in size order.

Consider how tools are stored in a workshop. You may imagine a shadow board, commonly used in manufacturing environments, where tool draws or walls may be made with a foam structure to hold the tools. Each

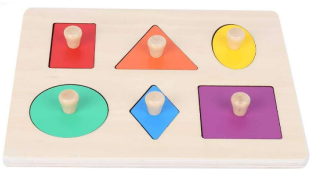


Figure 2: An example of the Shape Puzzle task

Figure 3: Example's of shadow boards

tool has its "shadow" cut into the foam so that every tool has a place and it is easy to see which tools are missing from the set, examples can be seen in Figure 3.

Now consider that once a job has been completed the shadow board has some tools missing, these might now be left on top of a workbench. We want a robot to place each tool back into its place on the shadow board i.e., to tidy the tools. We will constrain the problem to a single robotic agent (a statically mounted manipulator) and assume that the shadow board and tools are all within its reach. It should be immediately apparent that the tidying task is similar to the Shape Puzzle, albeit not necessarily with the ordering constraints placed on which item to insert. In this case though, the objects are more difficult to grasp.

A company might have hundreds of shadow boards with slightly different tool sets for different tasks, as tools are replaced over time they may not be replaced with an identical match, and over time also the tools will become scratched, dirty and worn.

The "Tool Tidy" robot has to perceive, identify, grasp and manipulate a set of objects with a variety of objectives e.g., handing tools which might have sharp edges safely to a person as well as holding them in an effective way to place them into the correct locations in the shadow board. The objects to handle are designed for human's to grip and hold ergonomically, and thus a fingered gripper may be more appropriate. It is also easy to see how this collaborative robot might be extended in capability to use some of the tools to assist the human, for example to perform fastening tasks with a screwdriver.

3.3 Beam Assembly

Light Guage Steel (LGS) framing describes a modern approach to modular building construction where structural components such as wall, roof and floor panels are made from beams typically pinned together with rivets or screws. The beams themselves are made through a process of bending and stamping sheet metal stock from a role. The beams have common connection designs but the exact specification of each beam is infinitely variable and drawn directly from architectural CAD software. Thus each beam and panel can be unique. Illustrative examples of an LGS panel assembly and of a building constructed with this technique can be seen in Figure 4a and 4b. Currently, the process of assembling these structures is to manually assemble 2D panels off-site in a warehouse where the work can progress regardless of weather conditions and in parallel with ground works to reduce total build timelines. Joints between beams include butt joints forming 90 degree corners and T's, angled butt joints, and entire sections of beams slotted through each other to form X structures at a range of angles. Once assembled 2D panels can then be shipped to the build site relatively efficiently, before connecting the pieces together into a 3D volumetric structure. In order to enable further efficiency gains in the building of these structures it is desirable to automate the off-site panel assembly process.

In order to make the "Beam Assembly" task practical for researchers we have designed a set of 3D printable joints which mount to 20mm square cross section aluminium extrusion. The collection of joint designs allow for beams to connect in a range of ways representing the types of conenctions seen in LGS structures including square corner and "T" right angle connections, connections at a range of angles, and connections where one beam passes through another. This kit allows for the recreation of LGS assemblies at a scale more amenable to the desktop robots seen in academic labs. Beams can be made through the combination of these parts and extrusion sections. These beams can then be built up into assemblies representing frame components. The initial component set can be used to make a range of assemblies of increasing complexity that contain multiple

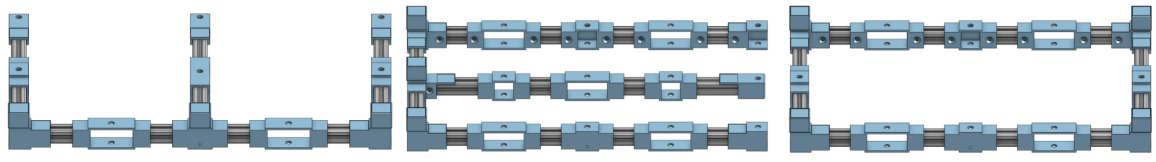
[1]Low cost LGS frame housing construction image from `https://www.howickltd.com/applications/rapid-building/low-cost-housing-system`

(a) An example LGS frame assembly

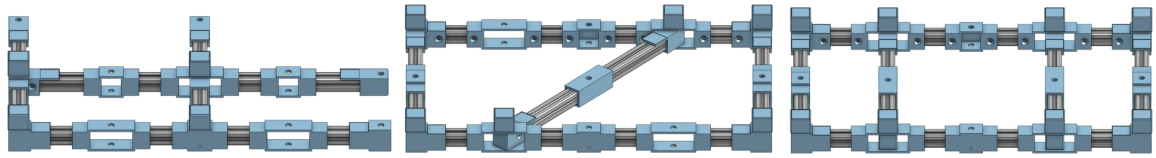
(b) LGS frame construction[1]

Figure 4: Light Guage Steel Frame Strutures

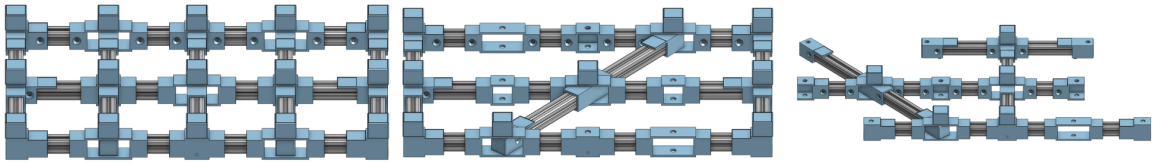
beam connection types reflecting the joints seen in the original LGS structures. A set of assemblies is shown in Figure 5 which have been published as the goal configurations of the RAMP benchmark [1].



(a) Easy Assemblies



(b) Medium Assemblies



(c) Hard Assemblies

Figure 5: Three classes of assemblies for assessing the capabilities of any proposed solutions to the RAMP benchmark.

To make the assembly task reversible we lock connections using a single pin at each joint. The assembly of full panels from the individual beams becomes an increasingly challenging task as more beams are added. Increased difficulty is driven by the multiple ways the beams can interact with each other requiring both one-to-one, and one-to-many object interactions.

We provide a standardised and easily extendable way to define beams and assemblies using specification files in xml format, an example xml file specifying a set of beams is shown in Figure 6a and an assembly specification in Figure 6b. These specifications allow us to extract relevant data automatically including assembly objects, parts of those objects and connections between them as well as dimensional data and information about visual markers which can be attached to our 3D printed parts to aid in beam localisation. We have created and published software for parsing these beam and assembly specification files which can automatically generate SE3 location data for final assembly positioning as well as domain information to enable reasoning about the assembly as will be discussed in Section 4.

```xml
<?xml version="1.0"?>
<data>
    <beam name="b1">
        <joint name="b1j1" marker="1" part="in-m-end">
            <child link="b111"/>
        </joint>
        <link name="b111" length="120">
            <parent joint="b1j1"/>
            <child joint="b1j2"/>
        </link>
        <joint name="b1j2" part="thru-m">
            <parent link="b111"/>
            <child link="b112"/>
        </joint>
        <link name="b112" length="120">
            <parent joint="b1j2"/>
            <child joint="b1j3"/>
        </link>
        <joint name="b1j3" marker="2" part="in-m-end">
            <parent link="b112"/>
        </joint>
    </beam>
    <beam name="b2">
        <joint name="b2j1" marker="3" part="in-m-end">
            <child link="b211"/>
        </joint>
        <link name="b211" length="120">
            <parent joint="b2j1"/>
            <child joint="b2j2"/>
        </link>
        <joint name="b2j2" part="thru-m">
            <parent link="b211"/>
            <child link="b212"/>
        </joint>
        <link name="b212" length="120">
            <parent joint="b2j2"/>
            <child joint="b2j3"/>
        </link>
        <joint name="b2j3" marker="4" part="in-m-end">
            <parent link="b212"/>
        </joint>
    </beam>
</data>
```

(a) Specification file for two beams each is a unique chain of links and joints.

```xml
<?xml version="1.0"?>
<assembly>
    <component beam="b7" base="True"/>
    <component beam="b4" />
    <component beam="b5" />
    <component beam="b3" />
    <component beam="b8" flipped="True"/>
    <connection name="C1">
        <element component="b4" joint="b4j1" />
        <element component="b7" joint="b7j1" />
    </connection>
    <connection name="C2">
        <element component="b5" joint="b5j1" />
        <element component="b7" joint="b7j5" />
    </connection>
    <connection name="C3">
        <element component="b4" joint="b4j3" />
        <element component="b8" joint="b8j1" />
    </connection>
    <connection name="C4">
        <element component="b5" joint="b5j3" />
        <element component="b8" joint="b8j5" />
    </connection>
    <connection name="C9">
        <element component="b3" joint="b3j1" />
        <element component="b7" joint="b7j3" />
    </connection>
    <connection name="C10">
        <element component="b3" joint="b3j3" />
        <element component="b8" joint="b8j3" />
    </connection>
</assembly>
```

(b) Assembly specification connecting 5 beams into a ladder structure.

Figure 6: Example beam structure description files.

3.4 Task Discussion

The tasks of completing the "Shape Puzzle" and tidying away the tools into their correct places are analogous to many packaging tasks in industry. For example seasonal products, short run exclusive products, or gift sets with combinations of products. Increasingly the task of packing these goods into sets is being delegated to logistics businesses as the producers themselves do not want to employ or retain a workforce of packing staff if they only produce goods that require packing like this seasonally. However, this means that the supply chain entity doing the packing task may not have access to part data such as object models. In order to complete these tasks the robot must have perception skills to find and identify the correct objects and manipulation skills to handle them. However, the assembly aspects of these challenges are relatively benign, with a limited amount of assembly constraints. The sequencing requirement of the "Shape Puzzle" increases the robot's requirement to be able to plan ahead but once the correct piece is found there are relatively few constraints on the assembly tasks.

Domains containing a large number of objects or physical states present challenges for efficient and accurate planning and control. The "Beam Assembly" tasks provides more challenging assemblies consisting of multiple interacting beams, with many components, assembly joint locations, and pins requiring the execution of long sequences of many challenging manipulation actions to complete each assembly. Actions such as the insertion of one beam through multiple others involve changing and multi-surface contact interactions.

Each task presents its own challenges for the robot; Table 1 highlights some of the sources of variation between the tasks. The following sections will describe a proposed domain independent framework for robot control applicable to all these tasks and many more.

Task	Grasping Modes	Object Classes	Task Actions	Sequencing
Shape Puzzle	✓	-	-	✓
Tool Tidy	-	✓	✓	-
Beam Assembly	-	-	✓	✓

Table 1: Sources of variation in the example manipulation tasks

4 Proposed Framework

4.1 Problem Formulation and Approach

The benchmark tasks present a set of challenging scenarios for a robotic agent. We wish to create an extendable framework for the control of a robot manipulator which provides an underlying structure for all of these tasks.

For all manipulation tasks there will be some similar components, such as a robot with the ability to move through space and carry objects, and others which are task dependent, such as the nature of the objects involved. We choose to build upon the general robot architecture REBA [3] which provides a two-layer architecture for tightly coupling AI reasoning and robot control, this has previously been shown to support robotic agents by integrating commonsense knowledge to increase task completion rates, and used for control of mobile robots to find and deliver objects. We contribute a general task-agnostic framework for manipulator robots building on REBA which can form the starting point for many manipulation agents with general a general set of skills such as robot motion, object grasping, and tool changing. For the benchmark tasks we also have developed task domains which supplement this robot domain in order to provide examples of how the framework can be applied.

Manipulation tasks can be specified by the desired motion of the task objects and assembly tasks are a subset of manipulation tasks focused on combining objects together to form a composite object. In order to build an assembly there are likely to be some ordering constraints governing which parts must be assembled first. Our beam assembly task is a good example of this where some beams must be assembled before others in order to avoid configurations where the ability to add a beam to the assembly has been blocked by previous additions. In simple tasks the sequence of assembly may be given; such as in the shape puzzle task. However, for more complex tasks it is advantageous for the agent to be able to work out a valid assembly sequence for itself. As such we implement an abstracted planning layer that takes object and task constraints in order to estimate a valid sequence of assembly. This abstracts away the robot domain, reducing complexity and increasing planning speed. The planned sequence steps are then used as goals to guide the robots planning and execution.

Just as a manipulation robot must be able to move though space, it must be able to interact with objects. A key interaction action for assembly robots is grasping, the forming of a relationship between robot and object, which allows the robot to hold and move objects. Each assembly domain may require its own set of actions based on the desired relationships between parts, but a general assembly robot will certainly require the ability to pick objects up. As such, we develop a grasping approach which is applicable to various grasp modalities and specifically which accommodates the integration of expert knowledge or previous experience to target grasps which maximise the probability of success for subsequent manipulation actions.

Figure 7 provides an overview schematic of our framework. The remainder of section will provide more detailed descriptions of the key components of the proposed general framework for robotic assembly applicable to a wide range of similar robotic assembly problems. In this section we will:

- Explain how we structure manipulation problems using Action Language $\mathcal{AL}_d$ [2] and Answer Set Prolog (ASP) [8] to form a multi-resolution reasoning agent.
- Outline our approach for Abstracted Task Planning to find assembly sequences that provide a sequence of achievable subgoals for the robotic agent to reason about.
- Detail construction of a location graph implementing next-to and near-to location distinctions as used to implement robot motion in our framework. Including an example of automatically generating this structure for beam assemblies.
- Outline our methodology for using task knowledge to support grasp selection to maximise probability of subsequent task success.

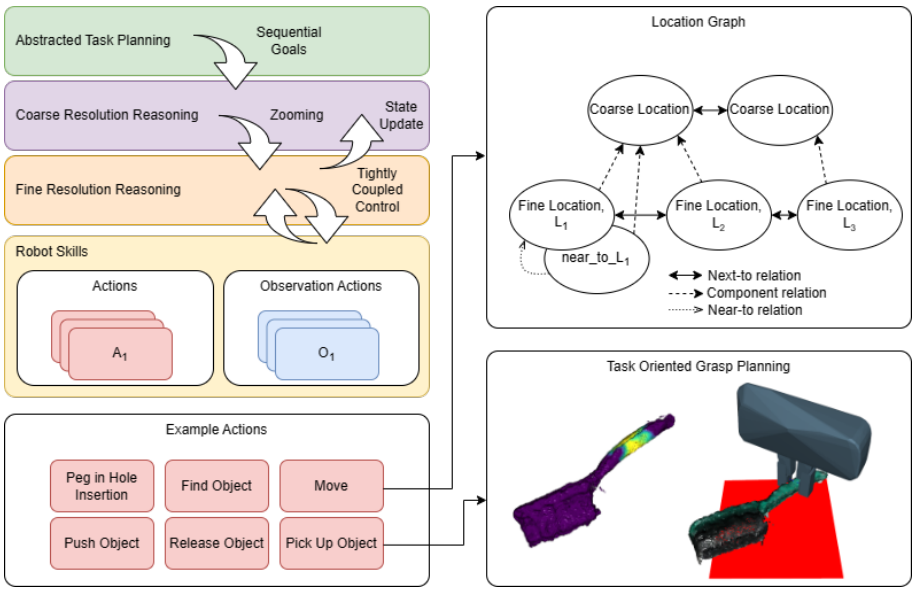


Figure 7: A high level overview of our general framework for robotic assembly, highlighting the flow of control and key low level skills.

4.2 The Manipulation Robot Domain

Logical statements provide a concise and structured way to reason about the relationships between objects. A valid assembly plan consists of a sequence of actions for the robot to build the target assembly. However, logical reasoning can rapidly become inefficient in large domains. If we model assembly tasks only at the resolution of the robot's action execution, then the planning problem quickly becomes intractable. To keep assembly problems tractable we implement a subset of the REBA method [3] which reasons at multiple resolutions and implements a structured approach to link these representations so that an agent is both able to reason about problems large enough to be of interest, containing many objects, and yet in enough fine-grained detail to perform complex tasks.

Any robotic agent must reason with domain specific information e.g., the details of objects in its environment, or task specific actions. Such a robotic agent must also reason about more general information regarding its basic capabilities e.g., the robot can move between locations, change tools, and has some basic actions to allow for interaction with objects. First we focus on this domain agnostic capability which will be applicable to all assembly task domains. Our robot domain contains a single robotic agent which can take actions to interact with other objects in the environment. By default we provide 3 locations at the coarse resolution, most industrial scenarios for assembly tasks will have an input region or regions where the parts are presented to the robot, and the robot will then bring the parts together into an assembly in a specific location be that a jig, fixture or flat table surface. We also include an intermediate surface for the robot to be able to place objects if it needs to re-grasp or sort through them. More detailed descriptions of these locations must be provided by the designer when specifying the domain task specific attributes.

We also define generic sorts for objects and grasp modes and fluents describing relationships between objects for example to describe the whether an object is held by the robot. Actions are defined which the robot can take such as moving between locations, picking up objects and putting them down, and changing between grasp modes; along with casual laws which describe the effects of actions and executability conditions which govern when actions can be performed, for instance limiting the robot to picking up one object at a time by prohibiting the robot from picking up an object at a timestep where the robot is already holding something in its grip.

We define a coarse-resolution system description $\mathcal{D}_c$ using statements in the action language $\mathcal{AL}_d$ [2] which describe a sorted signature and axioms governing domain dynamics. The signature of $\mathcal{D}_c$ includes a hierarchy of *sorts* such as *robot*, *place*, *thing*, *object*, and *step* (for temporal reasoning); actions such as $move(robot, place)$, $pick\_up(robot, object)$; statics, i.e., domain attributes such as $next\_to(place, place)$ that cannot be changed; and fluents, i.e., domain attributes that can be changed, e.g., $loc(thing, place)$ and $in\_hand(robot, thing)$. This signature is used to encode an abstract description of the domain. Given this signature, the axioms in $\mathcal{D}_c$

include *causal laws*, *state constraints*, and *executability conditions* such as:

$$putdown(R, O) \textbf{ causes } \neg\ in\_hand(R, O)$$
$$loc(O, Pl) \textbf{ if } loc(R, Pl),\ in\_hand(R, O)$$
$$\textbf{impossible } pick\_up(R, O) \textbf{ if } in\_hand(R, O)$$

A plan of abstract actions (for any given goal) can be computed by constructing a program $\Pi(\mathcal{D}_c, \mathcal{H}_c)$ in Answer Set Prolog (ASP) [8] that encodes the initial conditions, $\mathcal{D}_c$, a history $\mathcal{H}_c$ of prior observations and action executions, and helper axioms that direct the search for the plan of abstract actions. ASP encodes default negation and epistemic disjunction, and supports non-monotonic logical reasoning, an essential capability for robotics applications. Planning, inference, and diagnostics can be reduced to solving Π to generate answer sets (using an underlying SAT solver) that represent the beliefs of the agent associated with Π. To compute answer sets, we use the SPARC system [22], which (internally) uses the clingo solver [56]. Through this method for any given goal inference in the ASP program a plan of abstract actions can be generated allowing our robot agent to sequence actions to achieve its goal.

Locations are intentionally defined abstractly in this coarse description, we can think of them as regions or volumes of space, to allow for minimal descriptions to be used for a given task. We provide the robot with paths along which it can move between locations by specifying $next\_to(place, place)$ static relations between locations, this creates a "location graph" for the domain which will be described further in the following section.

If the entire system was modelled at the fine-resolution then quickly the size of the planning problem becomes inefficient or intractable to solve. As such we use the *zooming* paradigm from [3] to focus on the fine-resolution transition diagram relevant to a specific coarse-resolution transition of interest. Automatically cutting away parts of the fine-resolution diagram irrelevant to the action e.g., when moving between two locations the robot does not need to consider its grasping mode but it will need to consider the fine-resolution locations through which it must move.

The fine-resolution system description $\mathcal{D}_f$ is defined as a formal *refinement* of $\mathcal{D}_c$. It can be imagined as viewing $\mathcal{D}_c$ through a magnifying lens to discover structures that were previously abstracted away by the designer, e.g., grid locations within specific places or parts of specific objects. The sorted signature of $\mathcal{D}_f$ includes sorts for these new structures and those in $\mathcal{D}_c$, and actions, statics, and fluents described in terms of these sorts, e.g., $in\_hand(robot, object\_part)$. The axioms are refined in a similar manner, and additional *bridge axioms* link the attributes in $\mathcal{D}_c$ and $\mathcal{D}_f$. This leads to a notion of *weak refinement* that guarantees that any given abstract transition in the coarse-resolution can be implemented as a sequence of fine-resolution transitions. The specific steps of the refinement process are described in [3].

REBA provides a further *theory of observations* which allows the robot to execute *knowledge-producing* actions and update the values of *knowledge* fluents. We consider this theory in developing or framework, but we leave this theory's implementation and the use of observations to detect and explain failures for future work.

In this general robot manipulation domain we provide fine-resolution refinements of the coarse-resolution description, for example new locations which are components of the coarse-resolution regions. In each task we will use the generic positions "above_input", "above_table", and "above_output" to force the robot to move through clear space when traversing between the regions. At the fine-resolution locations will be defined by 6DOF poses specified for the robot end effector though the specific coordinate data is not needed for our logical representation. This also uncovers a new generic sort of *non_placement_locations* which are a sub-sort of location at which the robots actions are constrained by a new executability condition that prevents objects from being put down in mid-air.

$$\textbf{impossible } putdown(R, O) \textbf{ if } loc\_f(R, C), non\_placement\_location(C)$$

The full system description of the generic robot manipulation domain in both coarse- and fine-resolution is provided in Appendix A in SPARC syntax.

4.2.1 Location Graph

Design for Assembly guidelines encourage engineers to design parts to be assembled together from the same direction to reduce non-value added turnover actions in the assembly process [5]. However, a robust robotics

solution must be able to handle assembly actions from multiple accessible directions in the case that non-linear and non-vertical insertions cannot be removed by design. Even if the assembly actions can be limited to top down insertion real assemblies are likely to be 3 dimensional. As such the 2D grid representation used in the motivating examples of [3] is not sufficient to represent the object locations needed for specifying assembly actions.

In REBA's motivating examples a mobile robot agent is imagined traversing a map of locations that represent rooms at the coarse-resolution and grid cells within those rooms at the fine-resolution. A 3D voxel grid is the obvious extension of the 2D grid world which could be used to segment volumetric spaces. However, a voxel grid of useful density would introduce a large number of locations, mainly empty space, which would quickly become intractable in a logical domain and remains limited in the resolution of positions that it represents by the size of the voxels. In contrast, assembly tasks require precision placement of components potentially to sub millimetre location accuracy. $SE3$ matrix notation is ideal for representing accurate 6DOF locations with arbitrary resolution in continuous space, and useful mathematically for specifying target locations for both objects and the robot end effector. In this section a method for defining relevant 6DOF poses in the logical domain for planning assembly actions is described.

At the coarse resolution, the robot assembly domain is separated first into 3 regions; the input area, assembly area, and intermediate area. Whilst the concepts of input and assembly areas are clear, the intermediate area provides a place for the robot to store or manipulate objects if needed. For example, in the blocks domain, blocks are loosely stacked in a container in the input; the robot may need to remove some blocks from the input region to find the correct block for the assembly sequence. In that case, it is more efficient to place the picked blocks to known locations rather than dropping them back into the container should they then need to be retrieved at a later time.

The fine-resolution domain reveals more granular locations, we proscribe each object in the assembly a set of *assembly_location*, within the assembly area, including a *target_location* and *approach_location*. Additional assembly locations such as *through* or *prerotate* locations are added as needed for items which must be assembled in two or more steps such as in the beam assembly example task, these must be specified by the designer when considering the task description. Each item is also given an *input_location* within the input region. The intermediate area can be refined as needed for example with a number of locations on a table top or fixture locations e.g. for turnover operations. Fine locations may be flagged as *non_placement_location* to prevent the robot from placing objects at these locations. All assembly locations apart from the *target_location*s are labelled *non_placement_location* to limit the search space of possible actions once the robot begins assembling a set of parts.

We define assembly actions as a set of concrete actions which allow the robot to move an object between relevant assembly locations, these will be task specific actions that might be executed using very different control mechanisms specified by the designer. In abstract form, these assembly manipulation actions are similar to the general $move\_f(robot, loc\_f)$ action with an additional causal law that updates the $in\_assembly(object)$ fluent. We prohibit the general $move\_f(robot, loc\_f)$ action from reaching assembly target locations whilst holding things through an executability condition, shown below, to force the agent to select an appropriate assembly action.

$$\textbf{impossible } move\_f(robot, place) \textbf{ if } in\_hand\_c(R, T1),\ target\_location(place)$$

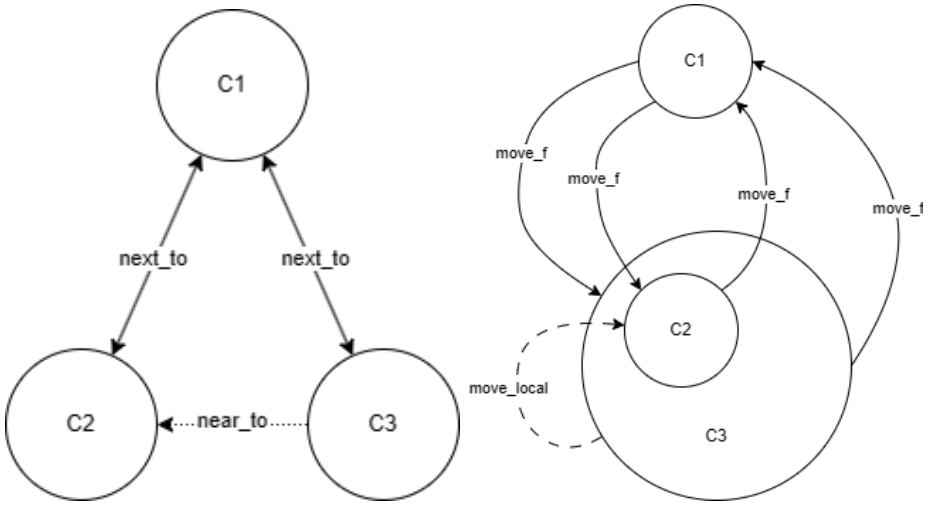
The next_to relation as defined in the REBA methodology can be reused in our new location structure providing the links between location nodes along which the robot can traverse the location graph by movement or assembly actions. However, some further restrictions are required on the actions that can be taken between locations. In REBA's two-dimensional grid world motivating examples locations were defined as regions of space that the robot or objects could be in, thus if an action failed the robot would either end up non moving, i.e., in the location which it started, or moving erroneously into an incorrect grid cell. Here we consider REBA's *theory of observations*, which allows the robot to take *knowledge-producing* actions and update the values of *knowledge* fluents, in this case detecting that it has moved to a different grid cell and updating the agents belief to reflect this new state.

In contrast, our new representation does not explicitly model all of the accessible space, instead it represents only important nodes in the task space. In this representation, if the robot attempts to move to a fine-resolution location but does not reach the desired $SE3$ pose with sufficient accuracy, perhaps because of collision or unexpected external forces, its new location may not be represented in our location graph.

In order to address this, an additional set of *near_to* locations, illustrated in Figure 8, are added to represent any position around a node in the location graph which is beyond a tolerance threshold where we might say that

the robot or object has reached the target location. An additional $move\_local(robot, loc\_f)$ action is defined so that the robot can recover, for example by using closed loop servo control to approach the target location from this *near_to* location. Whereas, typically a motion planning and trajectory execution approach is used in the $move\_f(robot, loc\_f)$ action. The *near_to* locations are also defined as *next_to* any locations which the target location is *next_to* to allow the robot to retreat and retry approach actions. Unlike *next_to* the *near_to* relation is not a two way relation between the locations. The decision to retreat or $move\_local(robot, loc\_f)$ will depend on the constraints of the movement actions defined with executability conditions, for example a task specific assembly action may require the robot to retreat and retry. The use of $move\_local(robot, loc\_f)$ is restrained by the following executability condition to prevent the robot from using this action to move larger distances where a standard movement approach is more appropriate.

$$\textbf{impossible}\ move\_local(R, C2)\ \textbf{if}\ loc\_f(R, C1),\ \neg\ near\_to(C1, C2)$$



(a) Relation graph of locations (b) Possible movement actions

Figure 8: $C3$ represents an automatically generated location *near_to* location $C2$ which is defined $next\_to(C1, C2)$. $C3$ encompasses a range of space around $C2$ beyond a nominal distance threshold.

Additionally, we propose a pruning heuristic that further improves reasoning performance by eliminating potentially many locations that are not of interest to the robot. This is achieved by reducing the set of relevant locations when performing *zooming* by eliminating locations which are a component of the coarse location, but are known to be associated with the assembly of non-relevant objects based on static axioms e.g., $target\_location(object\_a, fine\_location\_k)$. The *zooming* process defined in REBA already identifies a subset of relevant objects. In this example if object_a is not in the relevant subset, then we can also remove fine_location_k from our set of relevant locations as shown in Figure 9.

The SE3 representations of these locations, including rigid offsets for positioning the robot end effector e.g. for grasping or manipulation actions, can be calculated or extracted from observations in process. In this way a flexible and rich vocabulary for specifying locations in the logical domain is defined.

4.3 Abstract Task Planning

It is clear that considering the robots motion and behaviour adds a lot of complexity to the domain, introducing many locations and actions about which the robot must reason to successfully complete assembly tasks. In some industrial settings assembly tasks benefit from a defined sequence of part addition as in the shape puzzle example, but for more complex or flexible situations such as the beam assembly example there may be many plausible routes to completing an assembly.

In the baseline REBA coarse-resolution layer must consider assembly sequencing and robot behaviour simultaneously. It is possible to provide sequencing constraints by specifying relationships between the timesteps at which goals axioms are achieved, for example:

$$goal :- holds(assemble(rob0, O_1), n), holds(assemble(rob0, O_2), m), n < m.$$

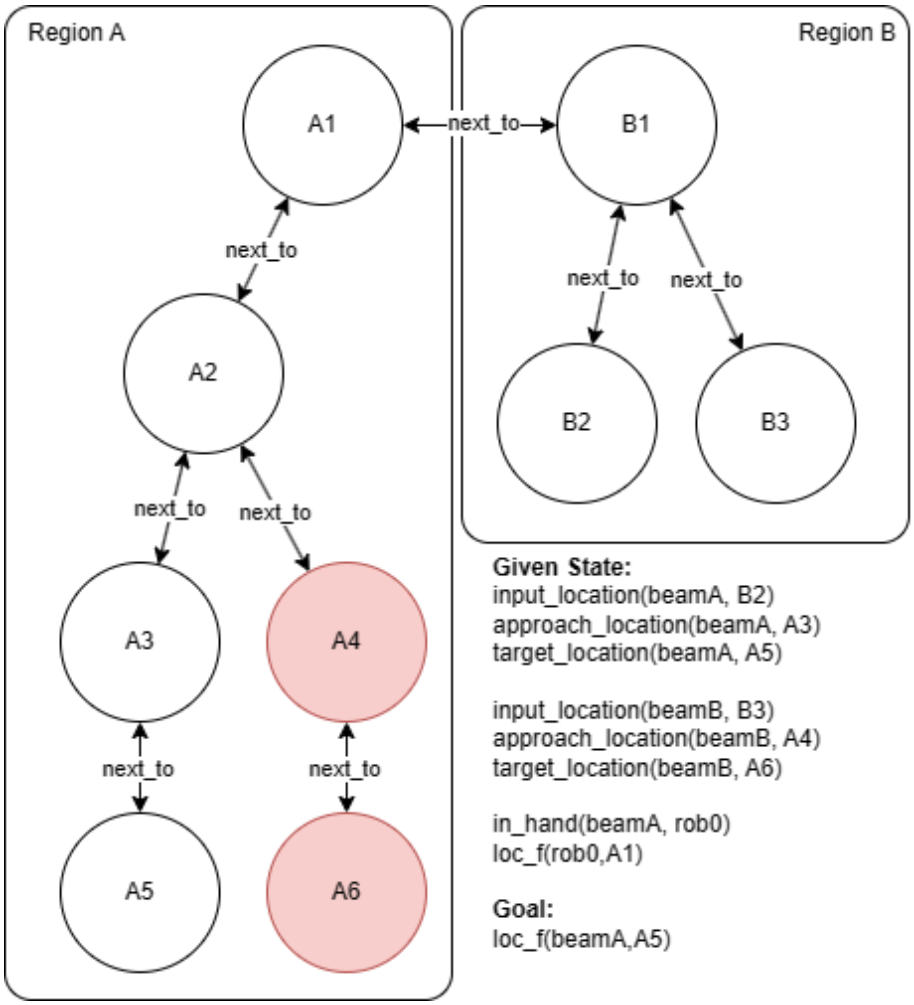


Figure 9: As more parts are added to the domain we automatically generate many nodes in the location graph for assembly of those parts, in this example when moving "beamA" we are able to prune the branch of the graph which is only related to "beamB" as this beam is not in the set of relevant objects. Locations within the coarse "RegionB" will already be removed from consideration by the *zooming* operation described in REBA as all states in the coarse-resolution transition maintain $loc\_c(rob0, RegionA)$.

However, this implies that some prior reasoning has provided this additional relation between n and m that gives the planner a sequence constraint. Constraints on the order or part addition create sub-goals towards the completion of an assembly task, significantly reducing the horizon of the planning problem i.e., if part O_1 must always be assembled before part O_2 then the robot can plan the assembly in two parts, from the first time step to step n to assemble O_1 and then from n to m to assemble O_2. Using breadth first search we expect to see the size of the search space grow exponentially by $O(b^d)$ where b is the branching factor and d is the depth of the solution in this case the number of time steps. Finding two short sequences should usually require a smaller search than finding one long sequence. Consider that the two shorter sequences should have the same length as the longer sequence in total, let's call the sequence lengths f and g, assuming that $b = f + g$, and a, b, c, d are all positive constants greater than 1 then:

$$\begin{aligned} b^d &= b^{f+g} \\ b^{f+g} &= b^f * b^g \\ b^f * b^g &> b^f + b^g \\ \therefore b^d &> b^f + b^g \end{aligned}$$

Thus we hypothesise that, for these more complex assemblies, finding a valid sequence of assembly and using that as a sequential succession of sub-goals for the robot planner will produce a more efficient planning approach.

The abstract planner must correlate precisely to the coarse-resolution reasoning layer in that any part added into the assembly in the abstract plan must represent a feasible sub-goal for the robot to attempt. As such we define the abstract task domain using $\mathcal{AL}_d$ as described for the robot domain, so that the state constraints and executability conditions are transferable into the coarse-resolution robot domain.

The abstract task domain signature must be provided by the designer for a given robot task and should include a single action *assemble*(*object*) with *causal laws*, *state constraints*, and *executability conditions* specific to the task, for example the in beam assembly task:

$$\begin{aligned} &assemble(O) \textbf{ causes } in\_assembly(O) \\ &supported(O1) \textbf{ if } in\_assembly\_c(O2),\ fits\_into\_c(O1, O2) \\ &\textbf{impossible } assemble(O) \textbf{ if } \neg\ supported(O) \end{aligned}$$

Here we have introduced two inertial fluents to represent the concepts of parts being added to the assembly, and parts being supported for addition if they have an assembly relationship with other parts already in the assembly such as $fits\_into(O1, O2)$. As well as an executability condition limiting assembly actions to supported parts only. Appendix B provides a full example abstract domain for a beam assembly task in ASP.

Given a starting state of a single part defined with a special sort *base*(*object*) that is the first part of the assembly, and assembly constraints between the parts, we can use the same ASP planning pipeline to find answer sets which meet the goal of all parts $P_0..x$ being $in\_assembly(P_x)$. We retrieve an assembly sequence by extracting the actions which occur at each timestep of a selected answer set:

$$occurs(assemble(O_1), 0), occurs(assemble(O_3), 1), ..., occurs(assemble(O_x, n).$$

Any valid answer set provides a possible assembly sequence for the robot to attempt, by searching for answer sets incrementally increasing the number of total steps, starting with a plan length of 1, we are able to identify the shortest valid assembly sequences. In our later experiments we will simply select the first such found sequence to pass to the robot as a series of sub goals, leaving to further work the analysis and selection of better or more reliable sequences of equal length or other more efficient methods of finding the shortest plans.

4.4 Combining Robot and Task Domains

The combination of the abstract task planner and coarse robot domain form the coarse-resolution robot-task domain. Here, the $assemble(O)$ action of the abstract domain is associated with a robot actor, becoming: $assemble(robot, object)$. The state constraints and executability conditions of the sub-domains otherwise combine without change with the exception of an additional executability condition added to constrain the agent to only assembling parts which the robot is holding.

$$\textbf{impossible } assemble(R, O) \textbf{ if } \neg\ in\_hand(R, O)$$

In some cases, elements of the generic robot domain may not be necessary for the task at hand, for example, in the beam assembly task, only parallel grasping is used and therefore multiple grasp modalities are not required. In this case, we advise that the designer remove any unnecessary elements of the generic domain in order to minimise the number of actions and objects for better performance.

In the abstract task domain we had ignored the physical interaction of robot and objects. Practically, when testing for the beam assembly task, we found that as the parts were not well fixtured, the assembly actions could knock the parts already in the assembly slightly, making future assembly actions much less likely to succeed. This is not an unlikely result in many assembly tasks where we seek to assemble larger structures. To address this, we add an additional inertial fluent to the domain, $misaligned\_c(object)$, which represents these inserted parts along with a corrective action $push(robot, object)$. An additional causal law is added to the assemble action and the push action is constrained with executability conditions:

$$\begin{aligned}
&assemble(R, O1) \textbf{ causes } misaligned_c(O2) \textbf{if } \neg\, base(O2), \\
&\qquad\qquad\qquad\qquad in_assembly(O2),\ O2! = O1 \\
&push(R, O) \textbf{ causes } \neg\, misaligned_c(O) \\
&\textbf{impossible } push(R, O) \textbf{ if } in_hand(R, T)\ thing(T) \\
&\textbf{impossible } push(R, O) \textbf{ if } \neg\, in_assembly_c(O)
\end{aligned}$$

Finally, the assemble action receives an additional executability condition which prevents the robot from trying to assemble objects when the assembly has become misaligned. This results in the robot having to correct the positions of disturbed items before proceeding to add more components to the assembly.

$$\textbf{impossible } assemble(R, B1) \textbf{ if } in\_assembly\_c(O2),\ misaligned\_c(O2)$$

We note that implementation of the probabilistic *Theory of Observations* from REBA could reduce the number of corrective actions applied by the robot by more accurately capturing the true state of the domain, sensing whether parts have become misaligned or not, and updating the robot's belief state.

4.5 Refining the Coarse Robot-Task Domain

The fine-resolution domain is a weak refinement of the coarse robot-task domain as defined in the original REBA paper [3]. We discuss the refinement process here using the beam domain as a motivating example.

The reader will remember that we describe the fine-resolution system description as viewing the coarse-resolution model through a magnifying glass, uncovering details which might previously have been abstracted away. The control mechanisms for robot actions can vary at the execution level but the fine-resolution domain must describe *concrete* actions which match the actions that the agent can execute. To construct the fine-resolution system description $\mathcal{D}_f$ first we construct a signature Σ_f by:

1. Preserving all elements of the coarse-resolution signature (including sorts, object constants, statics, fluents and actions)
2. Introducing new sorts s^* for each sort that is magnified consisting of components of elements of the coarse sorts s.
3. Introducing fine-resolution counterparts of each abstract function f magnified by the resolution change.
4. Adding static relations $component(O, O^*)$, which hold iff object O^* is a component of a magnified object O of sort s.

With this signature we can define the fine-resolution system description $\mathcal{D}_f$ by:

1. Replacing any abstract functions of the coarse-resolution description with their fine-resolution counterparts with variables ranging over appropriate sorts required by these counterparts.

2. introducing bridge axioms relating the coarse- and fine-resolution functions of the form:

$$f(X_1, ..., X_m) \;=\; Y \;if\; component(C_1, X_1),\; ...,\; component(C_m, Xm),\\ component(C, Y),\; f^*(C1, ..., Cm) = C$$

For further details and proofs we refer the reader to the original paper [3].

In our beam domain example we can consider the beams to be made up of modular parts, uncovering the new sorts *links* and *joints* which are subsorts of $beam_part$ a fine-resolution magnification of the coarse *beam* sort. Following the steps above we introduce the following component relation, abstract function and bridge axiom:

$$component(beam, beam\_part)$$
$$fits\_into\_f(beam\_part, beam\_part)$$
$$fits\_into\_c(B1, B2) \textbf{ if } fits\_into\_f(BP1, BP2),\; component(B1, BP1),\\ component(B2, BP2)$$

We also update relevant actions and axioms, such as adding a fine-resolution counterpart to the *pick_up* action and *in_hand* fluent, to reflect that in this resolution the robot is picking up *thing_parts* and that, in doing so, it now holds the relevant *thing_parts* at the fine-resolution, and *thing* of which this part is a component at the coarse resolution.

Furthermore each *joint* component of the *beam_part* sort is of a specific type defined by how it connects to other parts and different joints may require alternative assembly actions to put together. We go further than in previous REBA application examples by not only refining sorts, but also actions. At the coarse resolution we have the action: $assemble\_c(robot, beam)$, we refine this single coarse-resolution action into multiple fine-resolution actions to represent different concrete actions the robot must use to assemble the parts, for $assemble\_f\_square(robot, beam\_part)$ and $assemble\_f\_cap(robot, beam\_part)$ represent actions first, where the robot can insert a beam end-joint directly into a joint of another beam in the assembly and second, where the robot must place a beam to "cap" or encapsulate several beams already in the assembly by inserting it onto the previously positioned beams. These actions might involve very different control approaches or tuned behaviours at execution time and may have statistically different probabilities of success, therefore it is desirable to separate them in our agents reasoning model. Alternative assembly actions might include $assemble\_f\_through(robot, beam\_part)$ for joints where one beam passes though another, or $assemble\_f\_rotate(robot, beam\_part)$ for joints which are first inserted square and then rotated to a final position.

In practice we find that to split an action in the fine-resolution; first the designer should construct a fine-resolution description of the general action, a^*, in this case $assemble\_f(robot, beam\_part)$ following the previously outlined approach. Then create duplicate actions each with additional executability conditions to represent the newly uncovered fine-resolution state information which should drive the agent to choose between the variant fine-resolution actions. Each variant action must:

1. Only be applicable in a subset of the a^*'s starting states, but every starting state of a^* must be a valid starting state of one or more variant.
2. Not be applicable in any state at which a^* is invalid, i.e., all of a^*'s executability conditions must still apply.
3. The resultant state of the variant action (or a series of sub-actions) must be a resultant state of a^*, i.e., all of causal laws of a^* must apply.

In some cases a series of fine-resolution actions will represent a single action at the coarse level, this is expected as we seek to include more detail about the robot behaviour when reasoning at the fine-resolution. An example is when a beam needs to be inserted at an angle. It is feasible to specify this as a single action, or to separate the behaviours into two actions, to first insert the beam straight on, then rotate. In practice the second approach takes advantage of the existing ability to insert the beam square first which may be more reliable, the choice is up to the task designer. If a series of actions is chosen then suitable executability conditions and casual laws must be added to restrict the fine-resolution behaviour so that the path P of states in the fine-resolution domain represent the transition $T = \langle \sigma_1, a^c, \sigma_2 \rangle$ in the coarse domain and all states of P are extensions of σ_1 and σ_2.

4.6 Task Oriented Grasping

Given a general plan for manipulation actions such as assembly or handover tasks it is feasible for the robot to look ahead in the plan in order to obtain some understanding of the tasks in which a held object is likely to be used. This task-object relationship may require the robot to grasp the object in different ways. For example in the tool-tidy task the robot might be required to handover a spanner to a person, in this case the robot should hold the end of the spanner to present the handle to a person, but to use the spanner for tightening, the robot must hold the handle leaving the head of the spanner free so that it can be placed over the nut to be turned. Further more, holding the right part of the object is not sufficient to use the object efficiently but the correct grip type and position upon the object part are also important for maximising task-object mechanics.

In this section we present a method for training the robot to model task-object interactions as a function of finger tip placement, and a method using this learned function combined with other grasp quality indicators to generate better task relevant grasps in a way that is easily extendable to different robot hand designs. This section elaborates on the work published under the title **A Keypoint-based Object Representation for Generating Task-specific Grasps** presented as a conference paper at the IEEE 18th International Conference on Automation Science and Engineering (IEEE CASE2022) [4].

The reader will recall that given any example beam from the beam assembly task we model the beam object as a single sort for coarse-resolution reasoning, at the fine-resolution we uncover parts of the beam and classify them as link components or one of multiple joint component types. In order to successfully plan robot grasps on the beam we must go further and model the surface of the object that the robot will interact with. We want to place the gripper fingers onto the surface and generate a force closure grasp to trap the object in the robot's grip in a way that will allow the robot to apply wrenches to move the object or apply force through the object in order to achieve manipulation actions such as a contact rich assembly action.

To enable task oriented grasp planning we propose a three-level object representation aiming to closely link low level geometric and higher level abstract representations. At the top level, any object is an instance of a particular class (e.g., beam, cup, or hammer). The lowest level is a point cloud representation of an object's surfaces, encoding the geometric data (e.g., shape, size) necessary for planning grasps. The intermediate level is a small set of keypoints that concisely define the object's pose while allowing for in-class variations in shape and size which we link to the object-part representation of our fine-resolution domain.

4.6.1 Modelling The Object Surface

Low cost RGBD cameras offer an excellent tool for capturing the object's surface as a point cloud. These cameras have reduced in cost significantly over the last 10 years making them easily accessible to academia and typically combine time-of-flight or Infrared (IR) projection mapping technology with a colour camera to capture a single channel depth image alongside a standard three channel colour image of a scene. Typically, these images must be aligned in software as the lenses of the depth and colour cameras are physically separate, software libraries such as Intel's RealSense™provide open source implementations for depth image alignment. The view of an aligned RGBD camera can then be modelled as a rectangular pyramid called a view frustum using the pinhole model as show in Figure 10. A single camera position is only able to see one view of an object and low cost RGBD cameras are prone to shadowing near the edges of objects where depth data cannot be accurately recovered. In order to model the objects surface more fully and accurately we combine multiple views of the scene.

In order to combine multiple views of the object into a point cloud representation we use a method based on the probabilistic signed distance function (pSDF) [57], which allows multiple views of an object to be combined and models measurement uncertainty. We explicitly model measurement uncertainty in the captured representation in order to make use of this later when considering where to place the finger tips of the gripper on the objects surface. A voxel grid is used to capture the signed distance to the surface from the centre of each voxel, with distance data only stored for voxels close to the measured surface using a truncation threshold. Our implementation of the pSDF representation utilises a scalable truncated signed distance function (tSDF) meaning that, unlike the original implementation, the size of the tSDF voxel region is not constrained to a set number of voxels and can so can grow to accommodate a larger scene without loss of surface reconstruction accuracy. Instead of modelling distance uncertainty as a weight [58], surface distance is modelled as a random variable with a normal distribution $\mathcal{N}(\overline{d}_{sdf}, \sigma^2_{sdf})$. The effect of any new measurement $(d_{sens}, \sigma^2_{sens})$ at step k

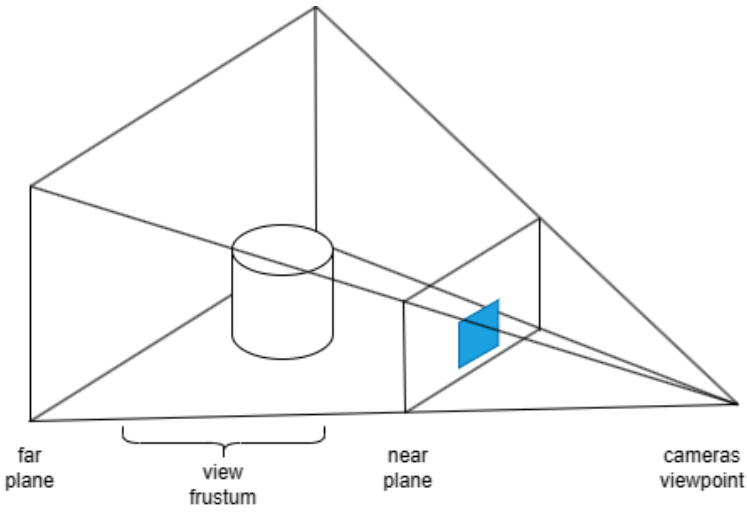


Figure 10: Illustration of a single cameras view of a 3D object, from a single viewpoint only one side of the object can be seen.

is merged using a Gaussian update:

$$\overline{d_k} = \frac{\overline{d}_{k-1}.\sigma^2_{sens,k} + \overline{d}_{sens,k}.\sigma^2_{k-1}}{\sigma^2_{k-1} + \sigma^2_{sens,k}} \qquad (1)$$
$$\sigma^2_k = \frac{\sigma^2_{k-1}.\sigma^2_{sens,k}}{\sigma^2_{k-1} + \sigma^2_{sens,k}}$$

where the measurement variance (σ^2_{sens}) is computed experimentally (see Equation 11). We implemented this encoding by revising the Open3D library [59]. A uniformly-sampled point cloud, with an uncertainty measure at each point, is then extracted from the pSDF representation using the marching cubes algorithm [60], Figure 11 shows a spirit level captured over three views from different positions, illustrating how multiple views of the same object can reduce uncertainty in the measured surface by colouring the extracted point cloud by uncertainty.

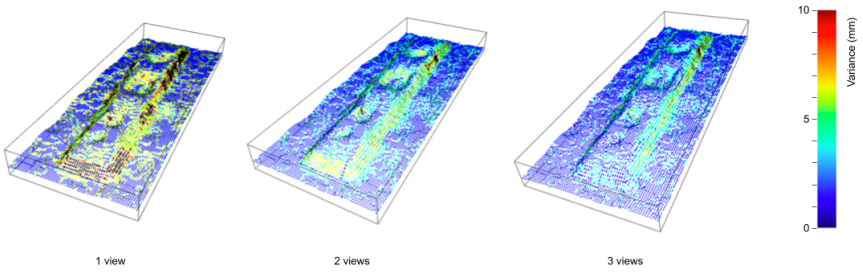


Figure 11: Pointclouds showing measurement uncertainty (σ^2_{sens}) update as multiple views are combined using the pSDF representation, three views are captured sequentially using a RealSense D415 camera.

It is necessary to separate objects from the background environment to target the correct object surface for grasping, Figure 13 shows recovered point cloud data of example objects using this approach. There are many ways to achieve this including simple plane removal for isolated objects on a flat surface or CAD matching to find known objects within a 3D scene. The designer will need to select an appropriate method for the manipulation task to be automated. Below we describe the method which we used in our grasping experiments which uses neural network image segmentation and might easily be applied to many other manipulation tasks.

We chose to segment relevant pixels from our input data before fusing views into our pSDF model, this reduces the amount of data in the pSDF update allowing for a finer voxel grid with a shorter update time as the number of calculations required to update the pSDF scales proportionately to the number of voxels. In our experiments, we used a the Detectron2 Mask R-CNN implementation from Facebook AI Research [61] to perform instance segmentation on the colour image data. Mask R-CNN is an extension of Faster R-CNN which performs object classification and bounding box detection, adding an additional branch which outputs an object pixel mask [62]. Transfer learning was used to train a Mask R-CNN model with a ResNet-50-FPN

backbone to segment a selection of example objects. Transfer learning reduces the training time and data needed for the network by using a pre-trained model which has been trained on a large dataset, in this case the COCO 2017 Object detection dataset [63]. In transfer learning we assume that the pre-trained network has learned how to extract salient image features, determine bounding boxes, and estimate object boundaries already. Rather than retraining the whole model from a default starting point with randomised weights; we instead only retrain the head of the model to tailor its outputs to find our objects of interest. Our example dataset contained approximately 500 images of our objects of interest on various background surfaces. Our object set contained 8 object classes with three or more instance objects of each class to provide in-class variation. We manually labelled about 350 images using the VIA online annotation tool [64] before switching to use V7 Labs [65] annotation software to speed up labelling of the remaining images, examples of labelled training images can be seen in Figure 12. We found that in addition to varying the background surface, using a torch to introduce lighting variation and using software data augmentation such as mirroring, random cropping and random rotation to increase the number and variety of images in the training dataset was useful in making the trained network more robust to the conditions in the workshop setting of our experiments. Training the network for 800 iterations took less than 15 minutes using a P100-GPU through Google colab[2].

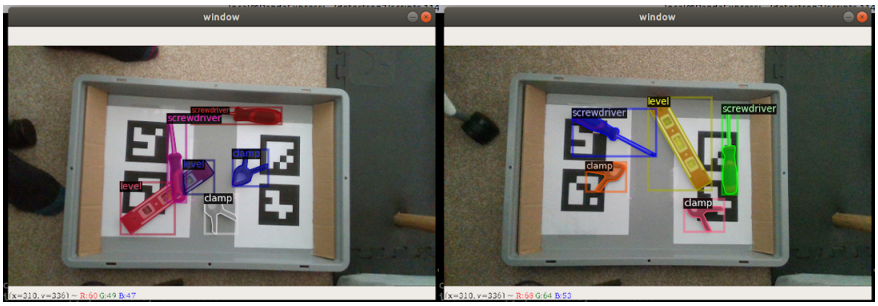


Figure 12: Examples of training images used to train Mask-RCNN instance segmentation

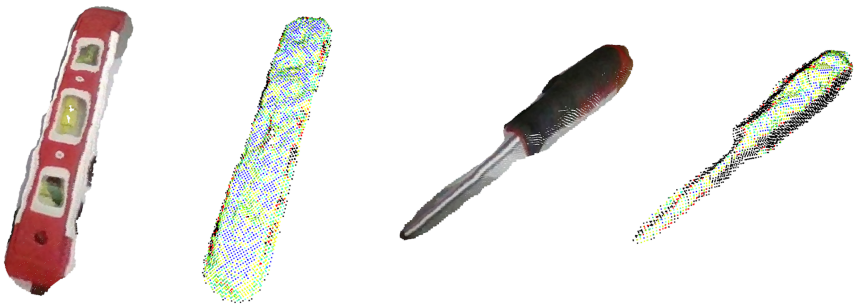


Figure 13: Recovered segmented object point cloud data

4.6.2 Keypoint Object Representation

We introduce an intermediate representation using a lightweight set of semantic object keypoints building on existing work [25, 26] which serves several purposes:

- Linking to our fine-resolution domain description of objects, links in the keypoint skeleton define object parts, and nodes define the boundaries between parts.
- Providing a lightweight object representation useful for encoding task-oriented grasping knowledge and planning manipulation motions through an object pose estimate in terms of the spatial relationships between keypoints.
- Modelling large in-class variations allowing us to automatically scale associated representations to new instances of a class despite changes in size, shape and orientation.

Figure 14(b) illustrates the keypoint representation for an example screwdriver class. In our implementation during data capture of multiple views of the object, 2D keypoints are detected in each colour image using a

[2] `https://colab.research.google.com/`

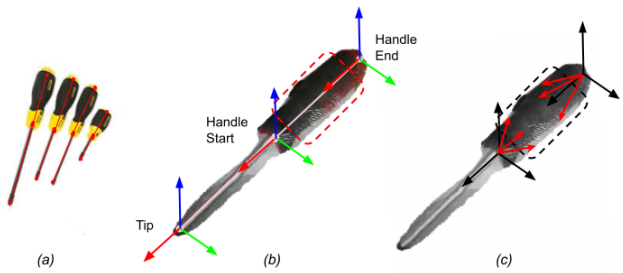


Figure 14: Keypoint representation: (a) illustrates in-class variation; (b) shows coordinate systems mapped to keypoints, a domain expert associates good grasps for fastening task with points in dashed region; (c) vectors between relevant keypoints and good surface points.

stacked hourglass network originally developed for human pose estimation [66]. We triangulate the 3D positions of keypoints from multiple views, with links between the keypoints forming a skeleton. This skeleton is used to guide grasp generation and to transfer knowledge for grasping across similar objects when used to perform similar tasks.

The stacked hourglass network was implemented in PyTorch and trained on labelled images from the tool-tidy example task. Each class required its own network as classes might have varying numbers of keypoints, changing the output dimensions of the network. In order to aid labelling, a simple python keypoint labelling tool was made to guide a user through a series of images saving the associated keypoint data into a yaml format which is easily loaded as training data[3]. The output of the network is a heatmap per keypoint to which we apply a gaussian blur to in order to remove extreme values before extracting the maximum point in image coordinates $P_i = (u_i, v_i)$, Figure 15 shows the extracted keypoints and mask for a screwdriver in a single image.

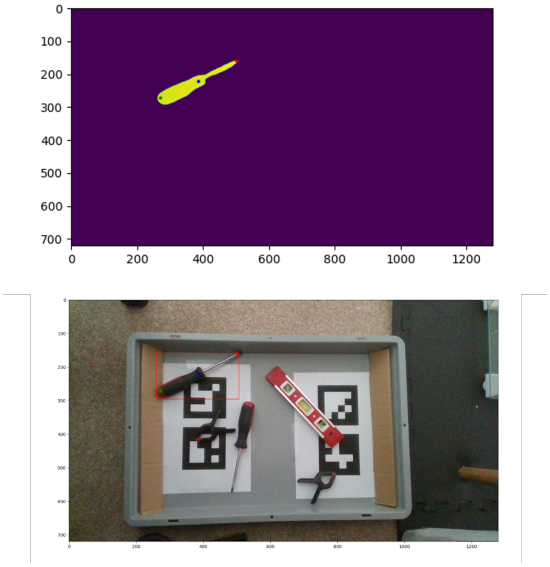


Figure 15: Keypoint and mask detection example for a single view of a screwdriver

For each view of the object we have a reasonably accurate model of the camera's intrinsic and extrinsic parameters through camera calibration. In order to obtain an estimate of a 3D position of the keypoint, we need to perform triangulation. However, due to small inaccuracies in calibration and limits of repeatability of keypoint detection from different angles, it is unlikely that the rays cast from camera origin to a detected 2D keypoint in each image will intersect. We use the midpoint triangulation method to estimate the 3D keypoint $P = (X, Y, Z)^\mathsf{T}$. For two images the midpoint triangulation method can be visualised as drawing the shortest line between the two rays and estimating the 3D point as the midpoint of that line, i.e., the found point minimises the euclidean distance to each ray. A generalised, closed form solution for midpoint triangulation of n rays is provided by Ramalingam et al. [67] given the start point of the ith ray A_i, and a unit vector B_i encoding the rays direction, $B = (B_1, ..., B_n)_{3\times n}$ and $C = nI_3 - BB^\mathsf{T}$:

$$P = \frac{1}{n}\{I_3 - BB^\mathsf{T}C^{-1}\}\sum_{i=1}^{n} A_i - C^{-1}\sum_{i=1}^{n} B_i B_i^\mathsf{T} A_i \tag{2}$$

[3]Keypoint labelling tool code is available from `https://github.com/M-A-Robson/keypoint_labelling_tool`

4.6.3 Encoding Task-Grasp Data

When performing robotic manipulation tasks, a key factor determining success or failure is whether the robot has grasped the object appropriately. Using our framework it is possible for the agent to look ahead, using the reasoning layers to form an idealised plan, before grasping an object to identify task constraints. If the robot does not grasp the object in a suitable manner for the task, it risks sub-optimal performance through additional re-grasping actions or poor task mechanics, and ultimately task failure. We propose a method for encoding functions which represent knowledge about how to grasp an object for a given task, using the keypoint skeleton representation.

We extend the keypoint representation to include a class-specific Euclidean coordinate system at each keypoint. Let a pair of keypoints be linked by a line segment which terminates at each keypoint and denote the length of this line segment, S. A euclidean coordinate system can be formed with the origin at one keypoint where the x axis is aligned to the link. The orientation of the y-axis and z-axis axes are defined by the plane formed by these two keypoints and one other keypoint, and the normal to the plane; if the object model has less than three keypoints, the coordinate system is based on eigenvectors of the object's point cloud.

Given the axes at a keypoint, unit directional vectors $\langle x, y, z \rangle = 1$ are computed to each recovered surface point of the exemplar grasp. These vectors are scaled onto a sphere of radius S and converted into spherical coordinates θ, ϕ where $\theta \in [-2\pi, 2\pi]$ and $\phi \in [-\pi, \pi]$:

$$S^2 = x^2 + y^2 + z^2 \tag{3}$$

$$\tan\theta = y/x, \quad \phi = \arccos\left(\frac{z}{S}\right)$$

The scaled vectors are used to construct a Gaussian Mixture Model (GMM) for each keypoint using an existing software implementation for automatically retrieving the optimum number of components with approximations of the Dirichlet Process inference algorithm and fitting using Expectation Maximisation [68]. To provide data points for learning the GMM we use an expert labelling approach where the task designer labels a single exemplar object surface with points in regions representing desirable finger tip placements labelled as good and all other points on the surface labelled as bad. An example of this approach and the GMM learned for a single keypoint in spherical coordinates is show in Figure 16. We use a mesh model as the exemplar object in order to extract evenly distributed points across the mesh surface accurately representing all of the part in 3D without noise or missing data which would typically be associated with RGBD data.

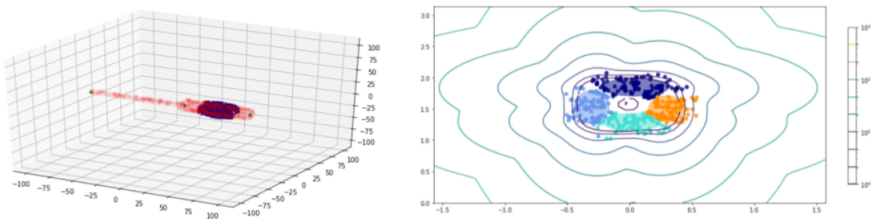
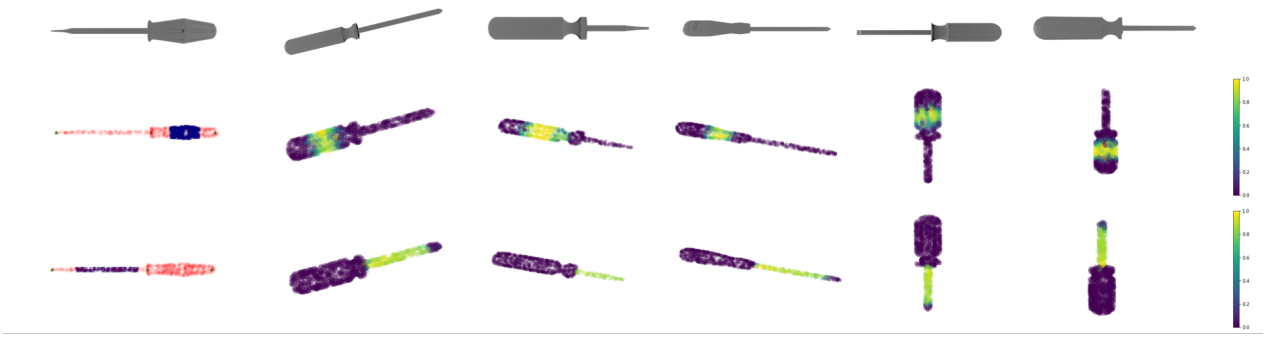


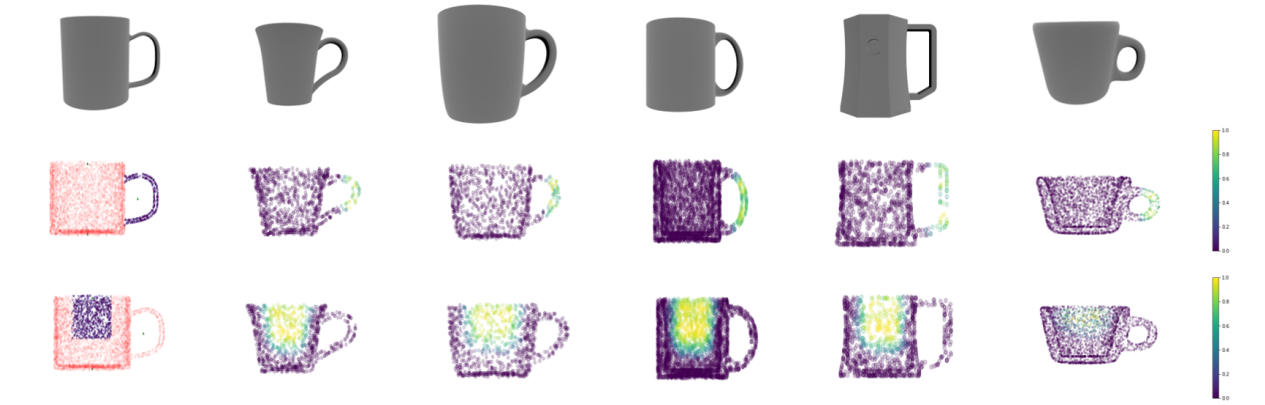
Figure 16: Right: Expert labelled grasp region for "tool use" task with a screwdriver object. Left: Negitive log likelihood contours of the learned Gaussian mixture model in spherical coordinates from the "'handle end" keypoint showing training data points colourised by GMM component.

Our representation based on spherical coordinates provides some robustness to scaling and orientation changes in new object instances of the corresponding class. At run time, point cloud points on the recovered surface of an object are assigned a probabilistic score based on the GMM models of the *relevant* keypoints, i.e., those that are on the nearest link (based on Euclidean distance). The object and task-specific probability value T_i for each surface point is calculated as the product of the probabilities from the relevant keypoints. Figure 17 shows the results of querying learned models, trained using a single labelled examplar, to score the surface points of new object meshes demonstrating that the models transfer well despite in class variation.

We note that many objects designed for manipulation by humans share similar features, a good example of this being "handle" parts. One advantage of our learned models is that the same model for grasping one object's handle may be intuitively transferred to a new object with a similar keypoint skeleton structure. We will return to exploring this concept further in our experiments.



(a) Screwdriver class *tool use* and *handover* task models.



(b) Cup class *tool use* and *handover* task models.

(c) Watering Can class *handover* and *tool use* task models.

Figure 17: Illustration of results of querying learned GMM's for new object meshes. The first column of each plot shows the mesh model of the examplar object and the labelled training data used to learn the GMMs, the remaining columns show a new instance mesh model and the recovered heatmap of scoring data for points on the new surface when we query the learned GMMs for each task.

4.6.4 Grasp Synthesis and Scoring

The objective of our grasp synthesis strategy is to identify good grasps balancing considerations of stability, measurement uncertainty, and task-specific constraints. There is existing work on considering different criteria to design and evaluate grasps with different grippers, e.g., estimating the marginal success probability $P(S)$ of a proposed grasp g, given object state x, as the sum of a weighted set of probabilistic criteria P_i [69]:

$$P(S|x,g) = \sum_{i=1}^{n} w_i.P_i, \quad \sum_{i=1}^{n} w_i = 1 \tag{4}$$

where w_i are the weights. We modify the original criteria to incorporate task-specific scoring functions and focus on factors relevant to any dexterous gripper. Additional functions and factors can be added to modify grasping behaviour for specific gripper designs or tasks.

One factor of interest is a good contact angle α between gripper finger contact vector and the surface normal of the contact region on the object. We estimate the probability of good contacts for grasps with any number k of finger tips:

$$P_1 = \left\{ \begin{array}{ll} \sum_{i=1}^{k} \frac{1 - \frac{2}{\theta}|\pi - \alpha_i|}{k}, & \prod_{i=1}^{k} z_i > 0 \\ 0, & else \end{array} \right\} \tag{5}$$
$$z_i = \left\{ \begin{array}{ll} 1, & |\pi - \alpha_i| < \theta/2 \\ 0, & else \end{array} \right\}$$

where θ is the static friction cone's maximum contact angle, which is set to 0.54 radians in our experiments.

The next factor is based on the insight that grasp success probability is dependent on the extent to which an object's surface is recovered from observed data. Poor surface recovery makes it difficult to estimate the surface location, and a surface with a high degree of curvature or variations may not provide a good contact region. The probability of a set of surface points being good contact points is given by:

$$P_2 = \left\{ \begin{array}{ll} \prod_{i=1}^{k} (1 - \frac{c_i}{c_{max}}).(1 - 3u_i), & c_i < c_{max} \\ 0, & else \end{array} \right\} \tag{6}$$

where u is a measure of the surface variation at each point based on the eigenvalue decomposition of points in its neighbourhood; $u = 0$ if all points lie on a plane and $u = 1/3$ if points are isotropically distributed [70]. Also, c is an estimate of the uncertainty in the location of the point, as obtained from the pSDF; c_{max} is the maximum value of this uncertainty, which is experimentally set to be 5mm.

Finally, each task and object-specific constraint is encoded as a function that assigns a score $\in [0, 1]$ to each recovered surface point describing its likelihood of being a finger tip location for grasps that meet this constraint. The overall probability for this factor is then the product of the values of probabilistic functions for individual constraints:

$$P_3 = \prod_j T_j \tag{7}$$

This model allows for the application of the previously described task-grasp knowledge models and is easily extendable with additional functions as desired by the task designer. For example in the beam task it is likely that we want to grasp near the centre of beams due to the rotational wenches caused by their significant length. An additional function could be incorporated into P_3 to penalise grasping the object at its extremities such as:

$$T_1 = \frac{2}{1 + e^{\frac{4.|d|}{d_{max}}}} \tag{8}$$

Where d is the euclidean distance from the point to the objects geometric centre and d_{max} the distance to the furthest point. This assumes that the object's centre of gravity is near to its geometric centre and that we desire a more stable grasp for our manipulation task. Both of these assumptions do not always hold and as such we may reason at a higher level that some feature of the object or task negates application of this constraint.

The grasp scoring system described above is applicable to any method that generates grasp candidates that can identify a set of proposed contact points on the object point cloud. For our experiments we implement a simple grasp synthesis algorithm which will be detailed in the remainder of this section.

The unified robot description format (URDF) provides a standardised approach to to describe the kinematic relationship between links and joints of the end effector or robot. Forward kinematics allows us to model the effects of joint actuation on a the robots links as an update to the sequence of rigid transformation chain locating the end links e.g., the gripper finger tips.

Letting $P_1, ..., P_n \overset{\text{iid}}{\sim} U(Z)$ where Z is the set of points extracted from the pSDF model using the marching cubes algorithm in the form:

$$Z_i = \langle X_i, Y_i, Z_i, N_i, D_i, T_i \rangle$$

Where N is the surface normal at the point, D is the point recovery score obtained using Eq. 6 and T is the task-object score obtained using Eq. 7. We generate a set of grasp candidates using 1 by iteratively closing a gripper defined by a URDF using eigengrasp notation to capture which gripper fingers are used and how the fingers should close relative to one another in the grasp [33]. We test for contact using the collision mesh for each link in *touch_links*, a specified set of links in the URDF that can form contact with the object, by checking after each closure iteration if any point of the point cloud is inside the boundary of the link's collision mesh polygon. We select a *primary_link* from the set of *touch_links* to place on the sampled surface in order to initialise the grasp candidate.

Instead of using a uniform distribution to sample the surface points for grasp candidate initialisation it is also possible to use the task score probability distribution to drive sampling towards task specific grasps. This is advantageous because a smaller number of samples are needed to get a good result, reducing processing time. A disadvantage is that this method tends to produce an unbalanced grasp where the primary finger has a better placement that the other fingers. To address this we use a subsequent local optimisation step for the best grasp candidates by iteratively offsetting the selected grasp candidate with small linear and rotational offsets before rerunning the closure algorithm 1. In our experiments we optimise only the single best grasp candidate using three iterations with initial offsets in grasp rotation and position of 15° and 5mm, with the offset values halved in each iteration.

Whilst the proposed grasp candidate generation method is clearly slower than some of the neural network driven methods used for parallel grasping we have chosen this approach so that the same method is applicable to a wide range of different gripper designs with different numbers of fingers through use of new URDF data and eigengrasp specifications. Alongside the flexibility of our grasp scoring method we believe that this is more widely applicable to the variety of manipulation challenges for which our framework may be applied.

Algorithm 1 Grasp candidate generation

Input : Hand H, Eigengrasps E, number of samples n
Output : Set of valid grasp candidates, G

Initialise Candidate Grasps $G = \{\}$
for *Sampled points* $P_0, ..., P_n$ **do**
  for *Eigengrasp eig in E, samples n, rotation angles R* **do**
    Position *primary_link* of H such that contact is formed at $\langle X, Y, Z \rangle$ in the direction of $-N$ with rotation about the contact axis r
    let $i = 0, contact = False$
    while $\neg$ *contact* $\wedge$ $(i <$ *max_iterations*$)$ **do**
      calculate Joint Positions J of H with $J = \sum_{i=0}^{b} a_i.eig$
      calculate Link Positions $L =$ **forward_kinematics**(H, J)
      for *touch_link h in H* ***except*** *primary_link* **do**
        $contact =$ **test_contact**(H, PCD)
      end
      $i = i + 1$
    end
    if *contact* **then**
      calculate contact quality q **using** Eq. 5
      calculate grasp score $g(d, q, t)$ **using** Eq. 4
      Grasp Candidate $c \longleftarrow \langle J, L, g \rangle$
      $G \cup \{c\}$
    end
  end
end

5 Experimental Results

In this section we collate the experiments conducted during the study to establish the viability of our framework and provide a discussion of the experimental results obtained. We experimentally test the following hypotheses:

H1 Our grasping method provides better grasps than baseline methods by balancing stability constraints with task/object-specific constraints;

H2 Our tiered object representation supports reuse of the learned task-specific object models for other similar objects being used to perform similar tasks;

H3 Our grasping method provides robustness to variations in size, shape, scale, and orientation within each object class.

H4 Our multilevel architecture with zooming allows us to plan action sequences for larger domains of object assembly than would be possible without layers of refinement.

H5 Our multilevel architecture with zooming reduces the overall planning time to generate a sequence of proposed actions for a given assembly.

H6 The addition of the abstract task planning level improves planning times and supports larger domains than the original REBA formulation.

H7 Our location pruning heuristic reduces planning times through a reduction in the number of objects in the zoomed domain during assembly tasks. The effect of pruning should be more pronounced on larger domains where more locations are pruned.

5.1 Grasp Planning Experiments

Hypothesis **H1**-**H3** were evaluated through implementation of parts of the tool-tidy task in the context of a robot assisting humans by grasping objects on a tabletop.

Physical experiments were conducted using a Franka Panda robot manipulator, with low-cost Intel Realsense D415 cameras mounted on the end effector, and statically in the workspace, as shown in Figure 18. Objects were placed on the gray tray to ensure it was within the robot's workspace and field of view. We segmented the objects from the background in each image as described in Section 4, and use the pSDF algorithm to obtain a point cloud.

The pSDF algorithm requires a model of measurement noise for each pixel in the input images. In our experiments we used 3 similar cameras, two statically mounted and one on the robot end effector, this simplifies our measurement updates as the same error functions can be used for each camera. However, the pSDF algorithm can be used with a variety of cameras provided that an estimate of measurement update variance is provided for each data source. We combine 4 different views of the object input region, one from each static camera and two views from different positions of the robot end effector camera, segmenting the object data as described in Section 4.6.1. The measurement update variance σ^2_{sens} in Equation 1 is estimated as the sum of two known sources of measurement error for the Realsense cameras, which we set to an X resolution X_{res} of 1280 pixels and a horizontal field of view (HFOV) of 65 degrees. The first source is the RMS error E_{drms} of depth measurement, which is the noise for a localised plane at a given depth d in mm [71]:

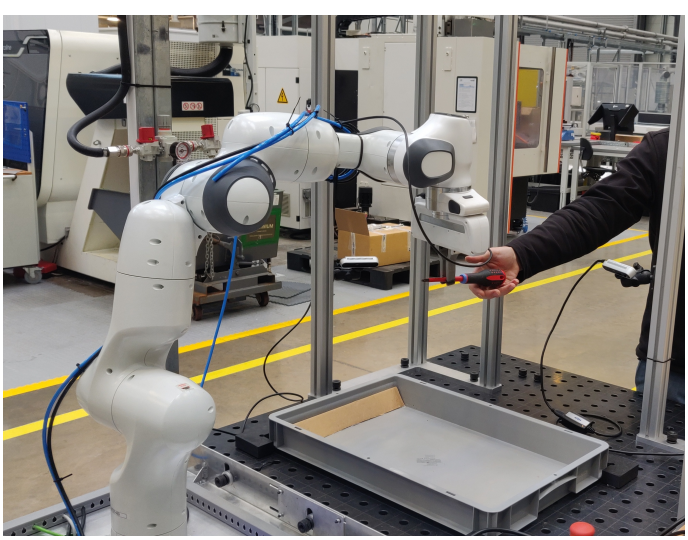
$$E_{drms} = \frac{0.08d^2}{55f} \tag{9}$$

where f is the camera focal length in pixels:

$$f = \frac{0.5X_{res}}{\tan\left(\frac{HFOV}{2}\right)} \tag{10}$$

The second source of error is based on the angle between the ray from the camera and the surface, measured as $\theta \in [0, \pi]$ radians. This error E_θ (mm) is estimated as described in [72]:

$$E_\theta = \frac{\theta}{(\frac{\pi}{2} - \theta)^2} \tag{11}$$



(a) Robot set up

(b) Object Set

Figure 18: Franka manipulator robot and some objects used in the experiments. Where to grasp an object is dependent on task-specific and object-specific constraints in addition to stability.

Object Class	Baseline Model			Handover Model			Tool Use Model		
	Stability	Handover	Tool Use	Stability	Handover	Tool Use	Stability	Handover	Tool Use
Brush	56.7%	52.9%	35.3%	83.3%	**100.0%**	0.0%	**90.0%**	0.0%	**100.0%**
Cup	**80.0%**	83.3%	12.5%	**80.0%**	**95.8%**	4.2%	76.7%	0.0%	**82.6%**
Dustpan	86.7%	**100.0%**	0.0%	**96.7%**	**100.0%**	0.0%	**96.7%**	0.0%	**100.0%**
Screwdriver	**83.3%**	12.0%	72.0%	80.0%	**91.7%**	0.0%	**83.3%**	0.0%	**96.0%**
Spoon	70.0%	52.4%	47.6%	**90.0%**	**88.9%**	11.1%	**90.0%**	0.0%	**100.0%**

Table 2: Summary of results from 450 trials split across 5 object classes. For each class, 30 practical trials spread evenly across three example objects (see Figure 18b) were conducted for each of three task-specific models (stability, handover, tool use). In each case, the "Stability" column indicates the proportion of successful grasps while the "Handover" and "Tool use" columns present the proportion of stable grasps which met the associated task criteria. Bold-faced numbers along each row indicate the best scores for the corresponding object class for each of the three tasks.

We considered six object classes for our physical robot experiments: cup, hammer, screwdriver, brush, dustpan, and spoon. Figure 18 shows the specific instances of each class used in our experiments. Object rotation was varied to create additional instances of each class, but cups were always placed with the opening facing upwards to allow the robot to grasp the handle.

For each class, we defined semantic keypoints (e.g., handle, top, bottom), exemplar grasps, and specific tasks. For object classes with a "handle" region, a grasp is suitable for the "handover" task if it leaves the handle unobstructed to allow another agent to grasp it when the object is presented. Each object class also supports a "tool use" or "pour" task. A grasp is suitable for the "pour" task if it leaves the outlet unobstructed to pour the liquid out. A grasp is suitable for "tool use" if it leaves unobstructed the region of the object that interacts with the environment when the tool is used, e.g., a hammer's head. These grasps should also target the handle that is often designed to maximise performance when performing associated tasks.

Hypotheses **H1** and **H2** were evaluated through robot trials using our method to pick different objects. We compared our method with a baseline method that does not encode task-specific knowledge, i.e., it considers $P1$ (Equation 5) and $P2$ (Equation 6) but not $P3$ (Equation 7) described in Section 4.6.4. The relative weights in Equation 4 were tuned experimentally. In each trial, candidate grasps were generated as described in Section 4.6.4. The robot executed the best grasp found using MoveIt to plan collision free paths to an approach position offset 10cm along the gripper z axis and approaching the target pose with a linear motion before closing the gripper to maintain a closure force of 70N (the maximum continuous grasp force of the Franka Hand gripper). From the target grasp pose, the robot attempted to lift the object 10cm from the table and hold it for 10 seconds; if it succeeded, the grasp was recorded as being successful and stable. Suitability of each stable grasp to any given task was assessed visually against the given exemplars.

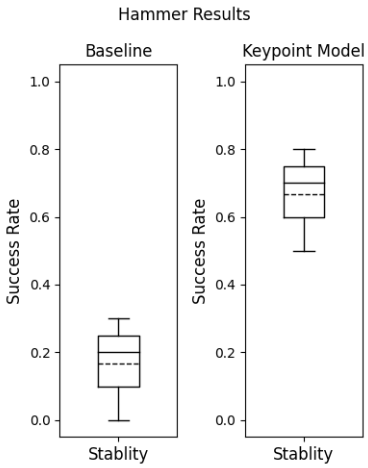


Figure 19: Trials in the hammer class focusing on stability; trials split 20:10:10 across the three hammer objects for each of the baseline and keypoint models for a total of 80 trials.

Hypothesis **H3** was evaluated qualitatively and quantitatively by exploring the use of task models' on images depicting a range of in-class variations. Our initial experiments on generating task-grasp fingertip placements on a range of previously unseen instances of the same class are shown in Figure17, but the robot trials on real objects allow us to better evaluate how well this works in practice. In Figure 17a, we see that the learned models of task and object-specific constraints scale well to new object instances despite variations in the shape of the handle or length of the screwdriver shaft. The nonlinear shapes of the cups and watering cans are modelled and considered when evaluating the suitability of new surface points as grasp points in Figures 17b and 17c, with different sets of points being preferred for the handover and pour tasks. These qualitative results demonstrate our method's ability to use models learned from a small set of exemplars to evaluate grasps for different tasks and guide grasping towards locations favoured by the domain expert.

We completed a total of 530 grasp trials on the physical robot platform. Table 2 summarises results for the five classes *Brush*, *Cup*, *Dustpan*, *Screwdriver*, and *Spoon*. The results for the object class *Hammer* are shown separately in Figure 19 that focuses on the stability criterion to highlight some interesting results that are discussed further below.

In Table 2, bold-faced numbers along each row indicate the best scores for the corresponding object class for each of the three tasks. For example, for the class *cup*, the 'Baseline' and 'Handover' model were equally good for providing good stability, while the 'Handover' model and 'Tool use' model (i.e., the task-specific models) provide the best performance for the corresponding tasks. Our experiments showed that for some object classes the baseline algorithm is more likely to produce grasps which suit one task, e.g., handover for cups and tool use for screwdrivers. This result is expected as these tasks require grasps which place the fingers on larger, lower curvature areas of the model that better fit the stability criteria optimised by the baseline approach. Results also indicated that including task and object-specific information in the learned models for different classes steered grasps towards regions that better suit the task under consideration and produced more successful grasps.

To evaluate **H2**, we focused on the brush and dustpan object classes, which have a "handle" configuration similar to that of the screwdriver; we modelled this configuration with a keypoint at each end. In our trials for these new object classes, we applied the same "tool use" model which had been trained for the screwdriver. Our results show that this produced grasps with both high stability and task suitability illustrating that our method can be used to translate learned task knowledge to other similar object classes when they are used to perform similar functions.

The results also supported **H3**. For example, experiments in the screwdriver class showed adaption to scaling changes by stretching the learned models based on keypoint positions to handle length (size) differences. Also, the trials for the cup class illustrated the ability of our approach to handle intra-class variations in appearance and shape, with each cup having different height, diameter, thickness, and handle design. Figures 17a and 17b show some variations in these two classes in the point cloud representation; additional qualitative results are shown in Figure 20.

For some objects there is a small drop in stability with our approach when our task-specific models result in the robot targeting regions that the baseline algorithm is less likely to target, such as when the robot attempts to grasp the cup handle for pouring. This is expected as the handle regions typically have reduced low curvature surface area for the gripper finger tips to contact. The drop in stability is balanced by the other constraints

in grasp scoring resulting in grasps with an acceptable stability which also meet the task requirements. As an additional test, we added a 6mm thick strip of foam to the handle of the cup with the thinnest handle and repeated an additional ten trials with the pour task model. With a larger surface area for grasping, the rate of successful grasps increased from 70% to 90% with our approach, and the proportion of these grasps that met the criteria for the pour task increased from 71% to 88%. Overall, these results support hypotheses **H1-H3**.

The results in Figure 19 for the hammer class highlight some issues of interest. Many grasp algorithms in literature aim for the centre of objects to increase the likelihood of a stable grasp. In the case of a hammer this heuristic is flawed as the functional design of a hammer requires concentration of mass at the head end. With the baseline model and the parallel gripper, it was difficult to find a stable grasp to lift a hammer without the hammer rotating in the grasp. In our tests the robot was not able to successfully grasp, lift, and hold the hammer for the required time in many trials. The use of our learned models showed a significant improvement in stability over the baseline method because our approach learned to guide grasping away from the handle and towards the head end of the hammer shaft for stability (and handover). The functional distribution of weight and our choice of a simple, parallel motion, two jaw gripper make it difficult to run trials successfully for a tool use task even with our learned model as whilst the robot often selects the correct grasp points on the handle it is unable to maintain a stable hold the object for the required time to meet our 10 second threshold. This could be addressed by using a more suitable gripper with more fingers that wrap around the the handle to provide reaction force against the hammers tendency to pivot out of the gripper.

5.2 Reasoning Experiments

Hypothesis **H4-H7** were tested by generating sequences of plans for a series of frame designs incorporating increasing numbers of beams and pins into a ladder structure using the RAMP benchmark beam set. Each task domain extends the general manipulation robot domain with three coarse locations *assembly_area*, *intermediate_area*, and *input_area*. The general manipulation robot domain provides the robot the ability to move between locations as well as pick up and put down objects. A more detailed description of the general domain is provided in Section 4.2. The domain contains a single manipulator robot which starts with its hand empty in a neutral home location *above_intermediate_area* which joins the assembly and input areas through $next\_to(P1, P2)$ relations. We define the following task domains, illustrated in Figure 21:

- **D1** - Containing 4 beams joined in a square with 4 pegs. Equivalent to the Easy-3 assembly of the RAMP benchmark.
- **D2** - Containing 5 beams with 6 pegs.
- **D3** - Containing 6 beams with 8 pegs. Equivalent to the Medium-3 assembly of the RAMP benchmark
- **D4** - Containing 7 beams with 10 pegs.

In these configurations each additional beam adds 2 more pegs to be inserted at the connection points. Each assembly was started with one of the long horizontal beams labelled as the base beam to initialise the assembly.

We provide additional task domain information by automatically extracting from xml descriptions the assembly objects defined as beams and pins at the coarse-resolution and with distinct beam-parts at the fine-resolution. We automatically generate an input location and assembly locations for each object as discussed in Section 4.2.1. We initialise all parts except the base beam at their input locations, with the base beam fixed at its target assembly location.

The combined domain provides the following actions for the robot:

- *move(robot, place)* - to move the robot end effector to a particular location.
- *pick_up(robot, part)* - to grasp a particular part with the robot's gripper.
- *put_down(robot, part)* - to release a particular part from the robot's gripper.
- *assemble_square(robot, joint)* - to insert beams where a specified joint fits into an existing joint in the assembly.
- *assemble_cap(robot, joint)* - to insert a beam such that one or more joints in the assembly fit into the beam.

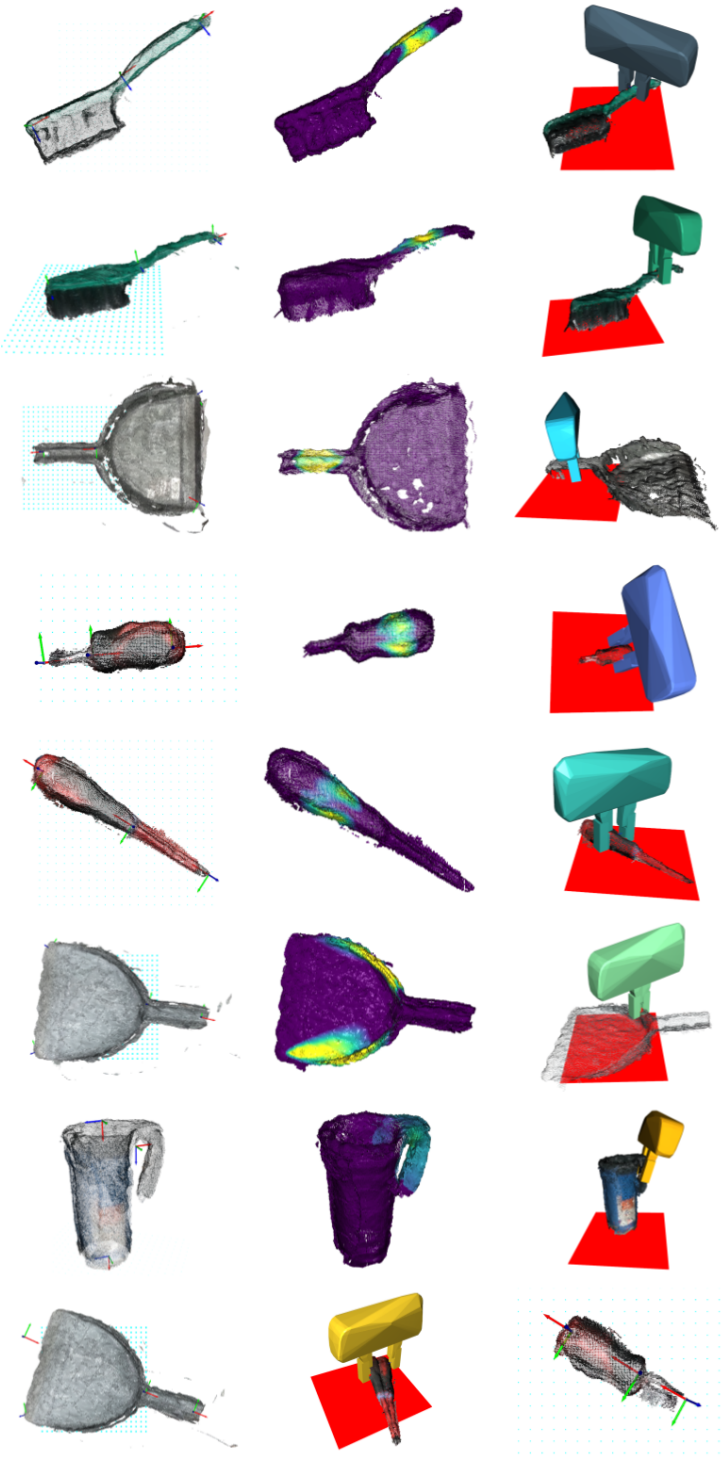
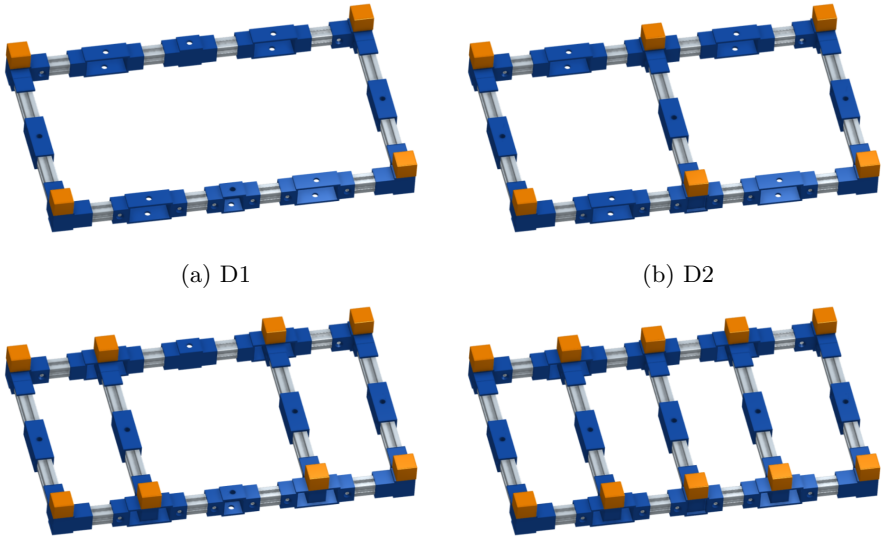


Figure 20: Illustrative examples of qualitative results using our grasping approach. The first seven rows show use of task-specific models to guide grasping despite intra-class variations in scale and orientation: first column shows keypoints; second shows heatmap of good grasp locations (lighter colours are better); and third shows preferred grasp. Rows 1-5 show use of same task-specific model (of *screwdriver* class) for the same task (tool use) across different object classes. Rows 6 and 7 show the "handover" task model for the dustpan class and the "pour" model for the cup class respectively. The final row illustrates some failure cases: firstly incorrect keypoint detection; second an instance where the object moved during imaging; and third an example of a point cloud with substantial noise, potentially leading to incorrect grasp placement.



(a) D1 (b) D2

(c) D3 (d) D4

Figure 21: Assemblies used for testing planning with increasing numbers of connecting parts, pegs are highlighted orange.

- *fasten(robot, joint, joint, peg)* - to insert a peg into two beams at their connecting joints.
- *push(robot, beam)* - to correct the position of beams already in the assembly.

We provide open source xml descriptions of each of these beam assemblies, the detailed domain descriptions, and all code used to run experiments via GitHub[4]. Experiments were run on a single desktop PC with a AMD Ryzen 7 3700X 8-Core Processor running at 3.59 GHz with 32.0 GB RAM. Graphical processing specification is not important in this case as the SPARC answer set solver is constrained to execution on a single CPU core.

First we conduct an experiment to track the execution times for each step of planning for domains of different sizes, for this experiment we use D1 and D3 as our example domains. We first attempted to plan in only the fine-resolution domain, starting with a step length of 0 and increasing the number of steps, $numSteps$, by a single step each iteration and recording the time taken for the planner to return a response. We set the goal to be the completed assembly, for example the goal definition for D1 containing beams b4, b5, b7, b8 and pins p1-4, where b7 is the base beam which starts in the assembly at $numSteps = 0$;

$$\begin{aligned} goal(I) \longleftarrow\ & holds(in\_assembly\_c(b4), true, I), \\ & holds(in\_assembly\_c(b5), true, I), \\ & holds(in\_assembly\_c(b8), true, I), \\ & holds(fastened\_c(b4, b8, p1), true, I), \\ & holds(fastened\_c(b5, b8, p2), true, I), \\ & holds(fastened\_c(b7, b4, p3), true, I), \\ & holds(fastened\_c(b7, b5, p4), true, I). \end{aligned}$$

A response from the planner takes the form of either an empty set or one or more answer sets representing a set of state transitions that include the goal at step I. We set a limit of 50 steps and a time out of 120 seconds per step. We then repeat the same process but this time using only the coarse–resolution domain.

Our results in Figure 22 show us that cycle time is affected by two factors:

1. The length of the plan i.e., the number of steps we are looking ahead.
2. The complexity of the domain i.e., the number of objects in the domain.

[4] `https://github.com/M-A-Robson/RAMP-Benchmark-Planner`

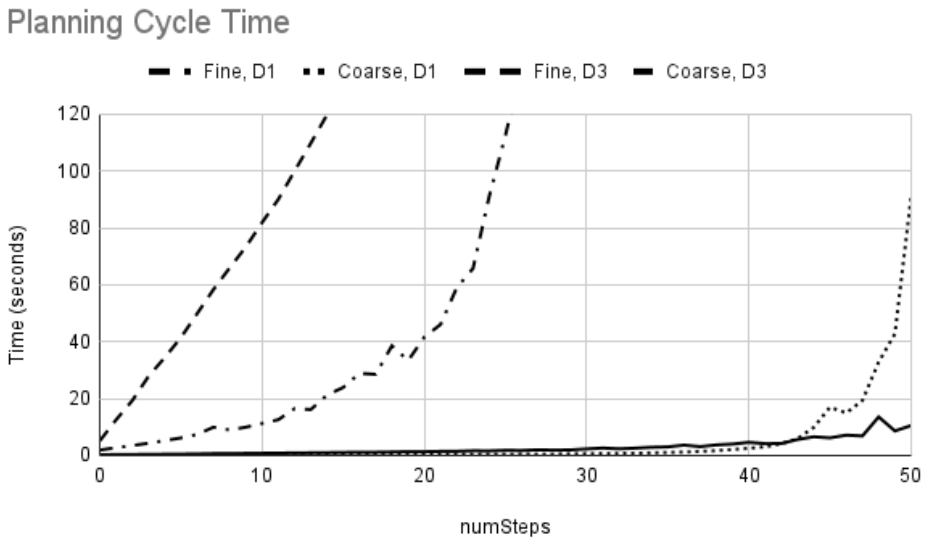


Figure 22: Cycle times for planning at different resolutions for domains D1 and D3, requiring 7 and 13 objects be added to the assembly respectively. Planning in the fine domain quickly becomes impractical with plans of greater than 25 steps timing out. The coarse planning times for the smaller domain spike drastically from 43 steps as the planner approaches the goal at 50 steps.

In essence we are performing a breadth first search considering the possible sets of states after each step where a single step represents a single possible action execution. Therefore, we expect to see the size of the search space grow exponentially by $O(b^d)$ where b is the branching factor and d is the depth of the solution (the number of time steps). All of the graphed results show an increase in cycle time as the number of steps increases. We can also see in the results that the fine resolution domains require significantly longer processing times for each step compared to the coarse domain equivalents. Using our iterative approach cumulative planning time is the sum of planning times per step, we can clearly see that planning only in the fine domain is not feasible for practical assembly tasks.

SPARC sets an internal limit on the size of answer sets to 100,000,000 characters so answer sets larger than this limit are considered intractable. We do not reach this boundary in this experiment but this provides the upper limit of the size of the search space.

We also see that cycle time appears to be affected by proximity to a goal state. This can be seen in the coarse-resolution planning cycle times for D1 which increase drastically from 43 steps as the planner approaches the goal at 50 steps. We speculate that this is due to the increased internal work that the SPARC system must perform to check if a goal has been reached and in parsing out the valid answer sets found when goals are achieved, in this case at step 50.

Next we compare our framework using an abstract task planning layer with the original 2-layer implementation from REBA [3]. We provide the same D1 domain each time. The abstract domain only considers the beams of the assembly and their relationships defined with statements such as $fits\_into(B1, B2)$ or $fits\_through(B2, B3)$ which are automatically extracted from the connection information in the assembly xml file. The base beam is still fixed at its target location as previously described. The abstract planner then searches for answer sets with a goal of the form:

$$\begin{aligned} abstract\_goal(I) \longleftarrow\ & holds(in\_assembly\_c(b4), true, I), \\ & holds(in\_assembly\_c(b5), true, I), \\ & holds(in\_assembly\_c(b8), true, I), \end{aligned}$$

The only action in the abstract domain is *assemble(beam)* as discussed in Section 4. Plans are found by the same iterative search method, starting with plans of length 0. When using the abstract planner the coarse-resolution planner is tasked with finding sequential goals based on the returned abstract assembly sequence. For example, if the sequence from the abstract planner returns a valid plan of the form:

$$occurs(assemble(b4), 0), occurs(assemble(b5), 1), occurs(assemble(b8), 2)$$

	Planning Time (seconds)			
Method	**Abstract**	**Coarse**	**Fine**	**Total**
REBA Baseline	0	267.90±0.92	33.81±0.16	301.72±1.04
Our Method	0.25±0.01	17.12±0.10	34.03±0.22	51.40±0.25
			Delta:	250.31

Table 3: Planning times averaged over 10 iterations of the D1 domain using the REBA coarse-fine planning approach and our extension using an additional abstract task planning layer.

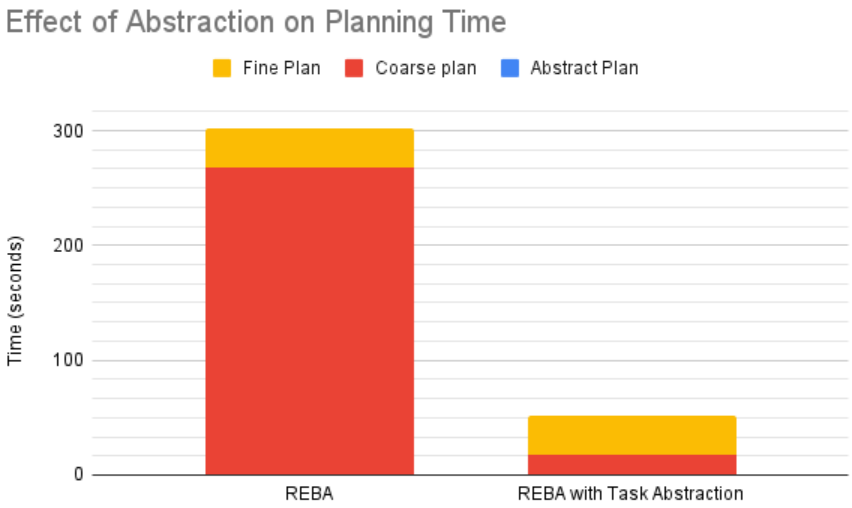


Figure 23: Graph showing the effect of the additional task abstraction planning step on the total planning time averaged over 10 iterations of the D1 domain, which requires the addition of 7 objects (3 beams and 4 pins) to the assembly. The abstract planning time is such a small portion of the overall planning time that it is virtually hidden in the graph, see Table 3

Then we provide the coarse-resolution planner with the subgoals such as;

$$goal(I) \longleftarrow holds(in\_assembly\_c(b4), true, I).$$

We then plan in the coarse domain until to find a valid sequence of actions that achieve the subgoal, recording the coarse state and history of actions to this point, then progressively feed the planner each subsequent subgoals until a valid coarse-resolution plan is found. We search for fine-resolution plans in the same manner as the original REBA paper, using the coarse-resolution states as goal definitions and zooming to only consider relevant parts of the fine domain. Table 3 shows the average planning times when applying our approach to the D1 assembly domain which are then illustrated in Figure23. Each table value is the numerical average and standard deviation over 10 runs. Our results show that finding valid assembly sequences in the abstract domain is very time efficient as our method of feeding sequential subgoals significantly reduced the time taken to find coarse-resolution plans. When using a starting plan length of 0 and performing an iterative search for a minimum length plan in the domain D1 the abstract planning step required only 0.25 seconds to find a valid assembly sequence for adding 3 beams, reducing total planning time compared to the REBA baseline by over 250 seconds. As expected fine resolution plan times remain roughly the same as before, benefiting from the domain size reduction of the zooming process from the original REBA paper. Both methods outperform planning in the fine domain alone which times out before finding a valid solution to the D1 assembly strongly supporting **H4** and **H5**.

In order to test applicability of our framework to larger domains we apply our search method to find plans for D2 - D4 which expand the domain by adding more beams and pegs to form more complex assemblies. Table 4 shows the increasing complexity caused by adding these additional components. Our method successfully finds plans for all of the test domains including D4 which requires 125 coarse-resolution actions that refine into 214 fine-resolution concrete actions for the robot to execute. Without the abstract planning step we are unable to formulate such long plans using the coarse-resolution planner as there are too many valid paths to the goal, resulting in answer sets over the 100,000,000 character limit that the SPARC library can successfully handle, and in-feasibly long search times. Planning times and standard deviations at each stage returned when using our method are recorded in Table 5 and illustrated in Figure 24.

Domain	n Beams	Assembled Beams	Pins: $2(n-2)$	Assembly Objects	Abstract Plan Steps	Coarse Plan Steps	Fine Plan Steps
D1	4	3	4	7	3	50	83
D2	5	4	6	10	4	74	125
D3	6	5	8	13	5	91	166
D4	7	6	10	16	6	125	214

Table 4: Planning data for the four test assemblies

	Time (seconds)				
Number of Objects Assembled	Abstract Planning	Coarse Planning	Fine Planning	Fine Planning, with Pruning	Pruning Delta
7	0.25±0.01	17.12±0.10	34.033±0.22	33.38±0.25	0.66±0.18
10	0.27±0.01	30.17±0.11	62.43±0.39	60.37±0.35	2.06±0.38
13	0.36±0.01	68.07±0.35	108.16±0.61	104.26±0.49	3.91±0.54
16	0.80±0.02	113.80±0.33	216.78±0.84	212.06±1.75	4.72±1.84

Table 5: Planning time results increase as the number of objects to be assembled grows, abstract and coarse data points are averaged over 10 runs with 5 runs each used to compare between fine planning with and without the pruning heuristic.

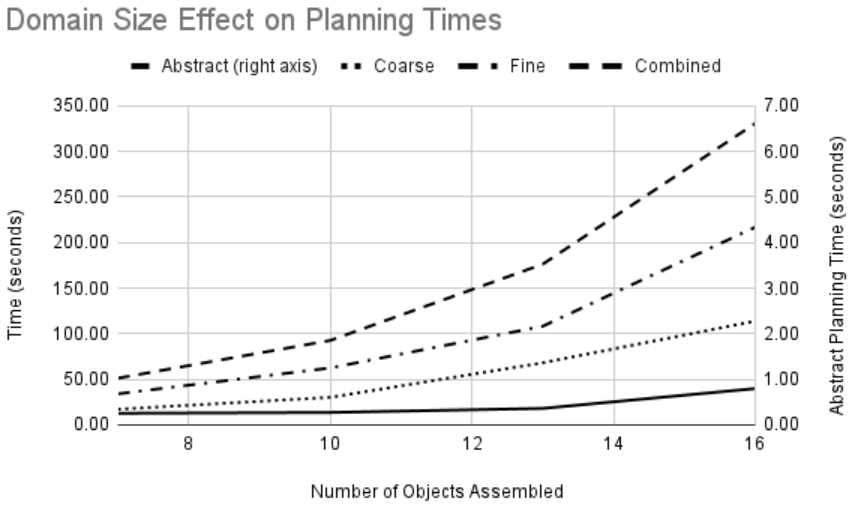


Figure 24: Planning times increase with the size of the domain.

Our results show that our planner scales effectively to these larger domains with the abstract planning step requiring an average of just 0.8 seconds to find a valid assembly sequence for adding 7 beams in domain D4 over 5 iterations. Considering the drastic effect that this has on reducing the planning time in the coarse planning step our results strongly support **H4**-**H6**.

We also test our location pruning heuristic discussed in Section 4. The heuristic appeared to have a small positive effect, reducing fine-resolution planning times marginally. Additionally, the average time reduction, shown in 25, does increase as the number of assembly objects in the domain grows confirming **H7**.

The effect of pruning in reducing the overall planning time was not as effective as we might have hoped. This may be due to the added overhead of additional processing of location data in the *zooming* step consuming almost as much time as is saved by reducing the size of the domain. Some location traversal sequences cannot benefit from pruning because they are not associated with robot moving a specific assembly object, limiting the application of the pruning heuristic. We note that *Zooming* already significantly reduces the size of the domain for planning fine-resolution actions representing transitions between coarse-resolution states. In general we found that these sequences of fine-resolution actions are not very long, averaging 1.73 fine actions per coarse-resolution transition across D1-D4. The remainder of this section will cover the results of applying our

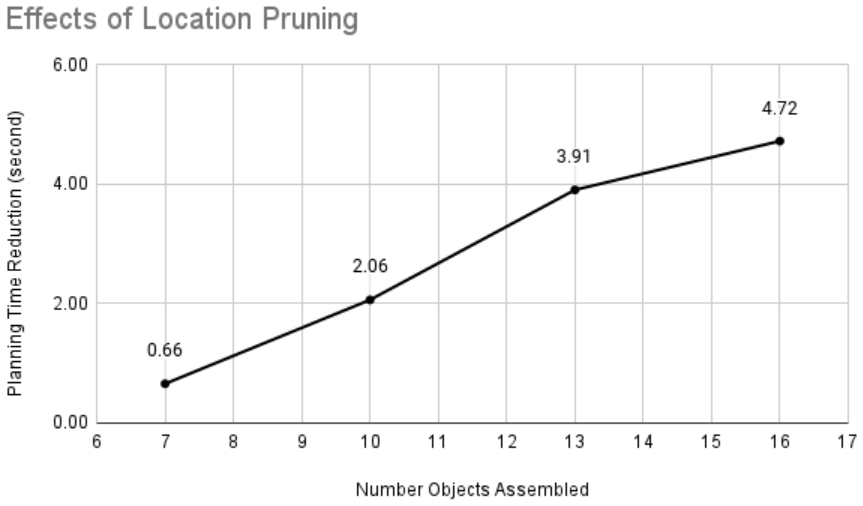


Figure 25: Time savings in fine planning step using pruning heuristic increase with the domain size.

framework to the Easy class of RAMP benchmark tasks. In these experiments the base beam is rigidly attached to the table and beams and pins are placed on a A2 paper template in order to lay them out in such a way that the robot can reach them, this template is available from the project website[5]. The implementation of concrete robot actions and running of trials on a physical robot were conducted by coauthors of the RAMP paper; Jack Collins and Jun Yamada of the Oxford Robotics Institute. We will outline the implementation details and trial results for completeness and to discuss the implications for our framework.

The baseline method presented in the RAMP paper [1] relies on our framework to find a successful plan before executing the found action sequence in open-loop using hand-designed concrete actions. Sensors (e.g., wrist-mounted and bird's eye cameras) are used to observe the April tags [73] on the beams and pegs, estimating their location in world coordinates. Since the offsets of these tags to each joint and link of the beams are known from our xml assembly description, joint and link poses can be computed in the global reference frame. We leverage the *MoveIt* [74] motion planner for executing the *move* action, and Cartesian-space impedance control and force feedback for contact-rich actions. The benchmark does not implement the grasp planner detailed in Section: 4, instead relying on predefined grasp poses relative to the sensed beam locations.

The *fasten* action is executed using a force based spiral contact search algorithm [75] with small oscillations used during insertion to prevent jamming. In the event of an unsuccessful hole detection, the robot moves upwards to re-estimate the hole position using the nearest fiducial marker observed by the wrist camera and re-attempts the insertion once more.

The *assemble_square* and *assemble_cap* use Cartesian impedance control to move the grasped beam towards the desired pose until a reactionary force, exerted against the robot, exceeds a threshold value. The *push* action is also implemented using Cartesian impedance control and is used to correct the position of a beam to a nominal target position in order to ensure alignment with future beams that might be inserted into the assembly.

We use the location system described in Section 4 to parameterise the motion and assembly locations in $SE3$ based on using the April Tags to locate the base beam and from this automatically determining the target and approach positions needed to complete the assembly actions of the other beams by the chaining of rigid transforms.

For each of the three Easy assemblies of the RAMP benchmark, shown in 5 five repeat trials were conducted with the results presented in Fig. 26. Fig. 27 shows snapshots of an assembly summarising the actions and showing execution of a single plan, further videos are available on the RAMP website. Fig. 26 depicts task completion percentage versus time (in seconds) across the three easy assemblies. Each subplot plots the average, best and standard deviation over the five repeats. On average the robot was able to complete 84% of the target assembly in an average execution time of 580 seconds. The standard deviation shown in 26 increases with execution time since the more complex assemblies typically require more actions and so the robot is more likely to fail in action execution.

[5] `https://sites.google.com/oxfordrobotics.institute/ramp/create-your-own`

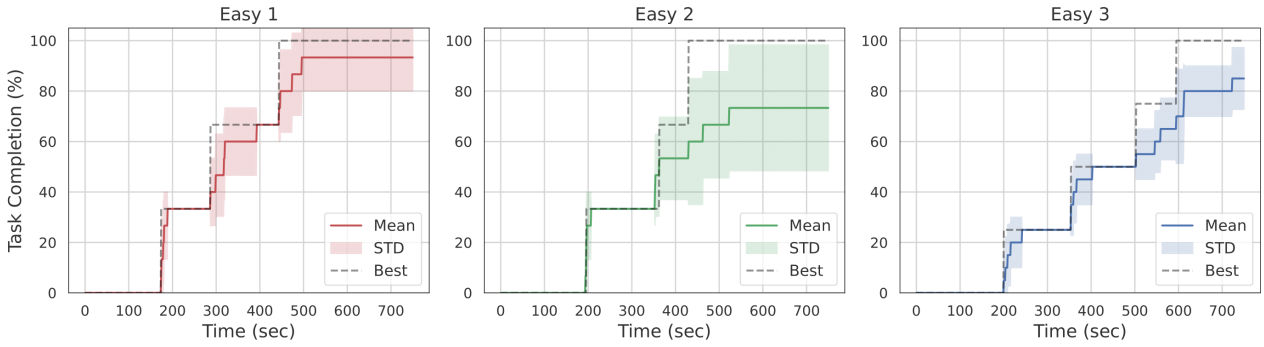


Figure 26: Results applying our framework to the three Easy tasks of the RAMP benchmark, with the percentage of completion of the desired assembly expressed as a function of the time (planning and execution, in seconds). The best and average of the repeats are plotted along with the standard deviation.

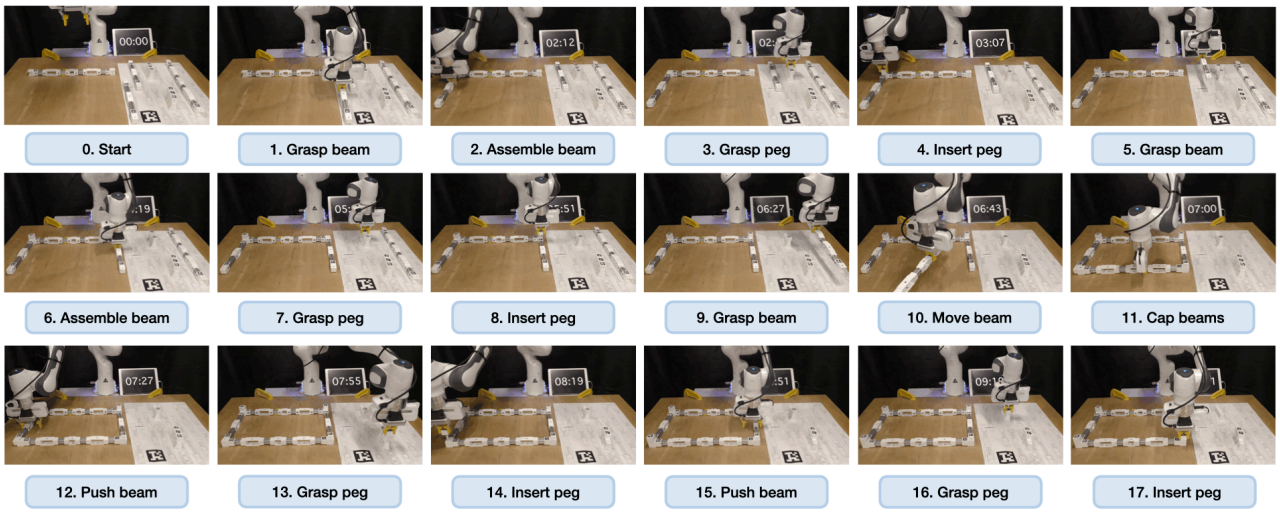


Figure 27: Visualising the assembly steps taken by the robot to assemble the RAMP Easy-3 assembly, equivalent to D1. Motion sequences have been shortened highlight object interaction actions.

In these experiments planning was conducted without the abstraction layer and takes 180-200 seconds, our results in Figure 24 suggest that the abstraction layer would reduce this by 50-75% for assembly domains of this level of complexity.

A significant portion of failure cases in the RAMP assembly experiments occurred due to failure of the peg insertion action. The fit between peg and hole requires a reasonable level of alignment and observation noise limits the accuracy of the robots pose estimations with real object. The hand crafted insertion skill relied on a spiral search which assumes that the peg is held vertically in the robots gripper which was not always the case. Furthermore, insertion of the peg into the designated hole is only possible if the two beams are well aligned as the peg must pass through matching holes in both beams of the connection. The peg insertion action could be made reliable through use of an external observation to check the pegs alignment in the gripper or by implementation of a learning-based method [76]. Fig. 26 shows clearly that the standard deviation on the assembly metric increases with time, implying that assemblies requiring a greater number of actions result in lower success rates, this is expected as each action execution carries a small probability of failure so over many actions there are more chances that the robot fails.

The curse of dimensionality causing an exponential increase in planning times is still apparent in the trend lines of Figure 24. Using our framework we are able to look ahead over the detailed assembly process in the fine domain of 16 or more objects, totalling over 200 concrete robot actions. The RAMP experiments on real hardware demonstrate that in practical applications, it is likely that the robot will struggle to execute this many actions without failure.

6 Conclusions

We have presented a general framework for the control of a robotic agent which has application to a wide range of assembly problems. The framework is built upon a tiered architecture consisting of an abstract task planning layer which feeds sequential goals to closely coupled coarse and fine resolution reasoning layers that direct a manipulation robot's behaviours,

The abstract task planning layer finds valid sequences of part addition to complete an assembly task given a task description specified using Action Language $\mathcal{AL}_d$ [2] and feeds these steps as a series of goals to a closely coupled robotic reasoning framework based on the REBA architecture [3]. The actions in the fine resolution domain description must be precisely matched to the concrete actions executable on the robot. Our results showed that the addition of the abstract task planner allows the robot to reason about action sequences over long time horizons needed for challenging assembly tasks such as the assembly of LGS beams in the construction of modular panels for the offsite construction industry. Our method demonstrated a 93.5% reduction in the total time needed to calculate a coarse-resolution plan for and example domain containing 7 objects to be added into a target assembly when using the abstract task-planning step compared to iteratively searching for a plan in the coarse-resolution domain alone. We showed that our method can scale to assemblies of practical sizes that would be intractable without the tiered architecture including finding sequences of over 200 robot actions in length to assemble structures requiring the addition of 16 objects.

A key part of our framework which enables assembly manipulation is a reusable location structure which can be used to automatically parameterise assembly behaviours in the logical domain and which directly translated into $SE3$ pose data suitable for robotic manipulation and assembly tasks. We demonstrated the use of this structure in parameterising the assembly actions of the RAMP benchmark tasks which represent a tabletop facsimile of the LGS beam assembly task. Additionally, we demonstrated a pruning heuristic to speed up searching in this location graph when the robot is searching for valid assembly motions.

We established that another key aspect of deploying a robotic agent to a manipulation domain such as assembly tasks is object grasping. Towards this end we also presented a grasp planning approach for considering task and object-specific constraints while generating suitable grasp points on the target object's surface for robot manipulation. We introduced a three-level representation for objects, compatible with our framework, which includes object class membership, point cloud data representing the objects surface, and semantic keypoints linked to the object parts. We presented a weighted grasp scoring model considering a minimum set of factors; a combination of measurement uncertainty and variation in the extracted surface, and the contact angle of gripper fingers to the surface. And demonstrated a method of learning a task-object model that preserves the relationship between keypoints and grasp points for specific tasks despite changes in factors such as the scale and orientation of objects.

We showed, through experimental evaluation on a Franka robot manipulator with a parallel gripper, that our grasping method was able to learn from a single expert labelled exemplar of an object class and apply this knowledge to generate grasps on previously unseen objects of that class which achieved the desired task-specific trade off whilst maintaining a high degree of grasp stability. Our experiments also provided promising results for the transfer of knowledge (models) learned for one object class to other classes that share similar semantic regions, e.g., handles.

7 Further Work

Our research opens up many directions for further work to address current limitations and explore interesting extensions. Testing the RAMP benchmark assemblies showed clearly that as the robot performs more actions using open loop control the likelihood of failure increases as each subsequent action carries a small risk that it does not change the state as intended e.g., the fasten action fails to insert a peg. The REBA architecture provides a *Theory of Observations* to address this by adding observation actions with probabilistic modelling of action outcomes so that the robot can reason about, detect, and react to these execution failures. The extension of our work to incorporate a similar closed loop control mechanism would provide an essential capability for the robot to be able to successfully complete long horizon assembly tasks reliably. Demonstrating application of the framework to other assembly tasks such as the example tasks which we used as examples to guide the framework's specification, and comparison of our approach to other baselines, such as PlanSys2 [77] which combines PDDL domains and behaviour tree execution control, would provide valuable insight into its advantages and limitations.

The general robot manipulation domain is currently constrained to a single robot in our applications, but specification using $\mathcal{AL}_d$ allows for expansion to consider several agents, enabling multi-robot collaboration which might be extremely useful for the building of larger assembly structures. Other authors have considered how multiple agents might use a similar reasoning framework to cooperate as an ad-hoc teams without prior coordination [78] which might be further investigated towards the development of assembly robots as a collaborating swarm.

Our reasoning framework allows the robot to look ahead to reason about the assembly actions which it must take at future time steps and to use this information to guide the planning of grasps to interact with objects in the environment. In our grasp planning experiments we explored the trade off of some task-specific criteria with stability; future work should expand on this to include additional object classes and criteria, for example many task-oriented grasps align with the object's principal axes and a measure of this alignment can also be considered when optimising grasp locations. Our grasping method specifically scores finger tip placement on the object but the orientation of the robots wrist and the object's relative pose in the gripper can have a large effect on the functionality of the grasp for task execution [43]. As such an interesting extension would be to include a model of approach vectors to relevant semantic keypoints to learn task-specific grasp approach vectors, in a similar manner to the finger tip placements, to learn more optimal configurations for task execution.

We provided our grasp planner with expert labelled exemplar objects from which to learn grasp points but it would also be interesting to reduce the extent of involvement of a human expert by exploring methods that automatically segment and process image sequences (i.e., videos) to learn semantic keypoints and grasp regions for additional objects. The overall objective would be to smoothly trade off different criteria to result in safe and successful grasps for a wide range of object classes and tasks. It would also be of interest to further explore the transfer of knowledge (models) learned for one object class to other classes with similar semantic regions. We were able to show that this concept worked for "handle" regions in our experiments but there is scope for further work to explore how far this concept is applicable to other object classes and other types of semantic region, e.g. packaging, doors and books all share hinging components that might have similar properties.

Finally, in order to fully combine the grasp planner into our framework we would like to build upon our work by testing the combined elements using the grasp planning element in tandem with planning and executing assembly tasks, in more practical and simulated example domains, to develop a truly flexible robotic assembly system.

References

[1] J. Collins, M. Robson, J. Yamada, M. Sridharan, K. Janik, and I. Posner, "RAMP: A benchmark for evaluating robotic assembly manipulation and planning," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, p. 9–16, Jan. 2024. [Online]. Available: http://dx.doi.org/10.1109/LRA.2023.3330611

[2] M. Gelfond and D. Inclezan, "Some properties of system descriptions of ALd," *Journal of Applied Non-Classical Logics*, vol. 23, no. 1-2, pp. 105–120, 2013.

[3] M. Sridharan, M. Gelfond, S. Zhang, and J. Wyatt, "ReBA: A refinement-based architecture for knowledge representation and reasoning in robotics," *Journal of Artificial Intelligence Research*, vol. 65, pp. 87–180, 2019.

[4] M. Robson and M. Sridharan, "A keypoint-based object representation for generating task-specific grasps," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 2022, pp. 374–381.

[5] G. Boothroyd, "Product design for manufacture and assembly," *Computer-Aided Design*, vol. 26, no. 7, pp. 505–520, 1994.

[6] M. Sealy and S. Corns, "Lucas design for assembly method applied at hawker siddeley switchgear," in *IEE Seminar on Team Based Techniques Design to Manufacture*, 1992, pp. 5/1–5/7.

[7] N. J. Nilsson, *Shakey the Robot*. SRI International. Artificial Intelligence Center, 1984.

[8] M. Gelfond and Y. Kahl, *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

[9] D. McDermott, M. Ghallab, A. E. Howe, C. A. Knoblock, A. Ram, M. M. Veloso, D. S. Weld, and D. E. Wilkins, "PDDL - the planning domain definition language," Yale Center for Computional Vision and Control, Tech. Rep., 1998. [Online]. Available: https://www.researchgate.net/publication/2278933_PDDL_-_The_Planning_Domain_Definition_Language

[10] S. Zhang and M. Sridharan, "A survey of knowledge-based sequential decision-making under uncertainty," *AI Magazine*, vol. 43, no. 2, pp. 249–266, 2022.

[11] I. Georgievski and M. Aiello, "HTN planning: Overview, comparison, and beyond," *Artificial Intelligence*, vol. 222, pp. 124–156, 2015.

[12] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *Journal of Artificial Intelligence Research*, vol. 61, pp. 215–289, 2018.

[13] J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer, "Search-based task planning with learned skill effect models for lifelong robotic manipulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE Press, 2022, p. 6351–6357.

[14] J. R. Anderson, M. Matessa, and C. Lebiere, "ACT-R: A Theory of Higher Level Cognition and Its Relation to Visual Attention," *Human–Computer Interaction*, vol. 12, no. 4, pp. 439–462, dec 1997.

[15] J. E. Laird, K. R. Kinkade, S. Mohan, and J. Z. Xu, "Cognitive robotics using the SOAR cognitive architecture," in *Workshops at the twenty-sixth AAAI conference on artificial intelligence*, 2012.

[16] D. Choi and P. Langley, "Evolution of the icarus cognitive architecture," *Cognitive Systems Research*, vol. 48, pp. 25–38, 2018, cognitive Architectures for Artificial Minds.

[17] M. Scheutz, T. Williams, E. Krause, B. Oosterveld, V. Sarathy, and T. Frasca, *An Overview of the Distributed Integrated Cognition Affect and Reflection DIARC Architecture*. Cham: Springer International Publishing, 2019, pp. 165–193.

[18] C. McGann, K. Rajan, H. Thomas, R. Henthorn, and R. S. McEwen, "A deliberative architecture for auv control," in *2008 IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 1049–1054.

[19] A. J. Coles, A. Coles, M. M. Muñoz, O. E. Savas, T. Keller, F. Pommerening, and M. Helmert, "On-board planning for robotic space missions using temporal pddl," in *International Workshop on Planning and Scheduling for Space*, 2019, pp. 34–42. [Online]. Available: https://api.semanticscholar.org/CorpusID:198994897

[20] M. Mayr, F. Rovida, and V. Krueger, "Skiros2: A skill-based robot control platform for ros," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 6273–6280.

[21] M. Colledanchise and P. Ögren, "Behavior trees in robotics and AI: an introduction," *CoRR*, vol. abs/1709.00084, 2017. [Online]. Available: http://arxiv.org/abs/1709.00084

[22] E. Balai, M. Gelfond, and Y. Zhang, "SPARC - sorted ASP with consistency restoring rules," *CoRR*, vol. abs/1301.1386, 2013.

[23] A. Brohan, N. Brown *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *arXiv preprint arXiv:2307.15818*, 2023.

[24] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser, "Tidybot: Personalized robot assistance with large language models," *Autonomous Robots*, 2023.

[25] W. Gao and R. Tedrake, "kpam 2.0: Feedback control for category-level robotic manipulation," *IEEE Robotics Autom. Lett.*, vol. 6, no. 2, pp. 2962–2969, 2021. [Online]. Available: https://doi.org/10.1109/LRA.2021.3062315

[26] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, "kPAM: KeyPoint Affordances for Category-Level Robotic Manipulation," in *International Symposium on Robotics Research (ISRR),*, Hanoi, Vietnam, 2019. [Online]. Available: http://arxiv.org/abs/1903.06684

[27] W. Zhou, B. Jiang, F. Yang, C. Paxton, and D. Held, "Hacman: Learning hybrid actor-critic maps for 6d non-prehensile manipulation," 2023.

[28] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis-A survey," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 289–309, 2014.

[29] D. Morrison, P. Corke, and J. Leitner, "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach," in *Robotics: Science and Systems*, 2018. [Online]. Available: http://arxiv.org/abs/1804.05172

[30] Y. Song, J. Wen, D. Liu, and C. Yu, "Deep robotic grasping prediction with hierarchical rgb-d fusion," *International Journal of Control, Automation and Systems*, vol. 20, pp. 243–254, 2022.

[31] Y. Yang, Y. Lui, H. Liang, X. Lou, and C. Choi, "Attribute-based robotic grasping with one-grasp adaptation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[32] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Dafle, R. Holladay, I. Morona, P. Q. Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez, "Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching," 2017. [Online]. Available: http://arxiv.org/abs/1710.01330

[33] M. Ciocarlie, C. Goldfeder, and P. K. Allen, "Dexterous Grasping via Eigengrasps: A Low-dimensional Approach to a High-complexity Problem," *Robotics Science and Systems*, 2007.

[34] Z. Deng, B. Fang, B. He, and J. Zhang, "An adaptive planning framework for dexterous robotic grasping with grasp type detection," *Robotics and Autonomous Systems*, vol. 140, 2021. [Online]. Available: https://doi.org/10.1016/j.robot.2021.103727

[35] E. Arruda, J. Wyatt, and M. Kopicki, "Active vision for dexterous grasping of novel objects," in *IEEE International Conference on Intelligent Robots and Systems*, 2016, pp. 2881–2888.

[36] E. Arruda, C. Zito, M. Sridharan, M. Kopicki, and J. L. Wyatt, "Generative grasp synthesis from demonstration using parametric mixtures," in *RSS workshop on Task-Informed Grasping*, 2019. [Online]. Available: http://arxiv.org/abs/1906.11548

[37] M. Kopicki, R. Detry, M. Adjigble, R. Stolkin, A. Leonardis, and J. L. Wyatt, "One-shot learning and generation of dexterous grasps for novel objects," *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 959–976, jul 2016.

[38] Q. Lu and T. Hermans, "Modeling Grasp Type Improves Learning-Based Grasp Planning," *IEEE Robotics and Automation Letters*, vol. Pre-print, pp. 1–8, 2019.

[39] A. Holladay, J. Barry, L. P. Kaelbling, and T. Lozano-Perez, "Object placement as inverse motion planning," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3715–3721, 2013.

[40] Z. Li and S. S. Sastry, "Task-Oriented Optimal Grasping by Multifingered Robot Hands," *IEEE Journal on Robotics and Automation*, vol. 4, no. 1, pp. 32–44, 1988.

[41] A. M. Ghalamzan E., N. Mavrakis, M. Kopicki, R. Stolkin, and A. Leonardis, "Task-relevant grasp selection: A joint solution to planning grasps and manipulative motion trajectories," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, oct 2016, pp. 907–914.

[42] F. Zacharias, *Knowledge Representations for Planning Manipulation Tasks*. Springer, 2012, vol. 16.

[43] A. H. Quispe, H. B. Amor, H. Christensen, and M. Stilman, "Grasping for a Purpose: Using Task Goals for Efficient Manipulation Planning," 2016. [Online]. Available: http://arxiv.org/abs/1603.04338

[44] D. Berenson, "Constrained Manipulation Planning," Ph.D. dissertation, Carnegie Mellon Univeristy, Pittsburgh, Pennsylvania, 2011.

[45] Y. Laili, Z. Chen, L. Ren, X. Wang, and M. J. Deen, "Custom grasping: A region-based robotic grasping detection method in industrial cyber-physical systems," *IEEE Transactions on Automation Science and Engineering*, 2022.

[46] B. Wen, W. Lian, K. Bekris, and S. Schaal, "Catgrasp: Learning category-level task-relevant grasping in clutter from simulation," in *ICRA 2022*, 2022.

[47] K. Qian, X. Jing, Y. Duan, B. Zhou, F. Fang, J. Xia, and X. Ma, "Grasp pose detection with affordnace-based task constraint learning in single-view point clouds," *Journal of Intelligent and Robotic Systems*, pp. 145–163, 2020.

[48] M. Kokic, J. A. Stork, J. A. Haustein, and D. Kragic, "Affordance detection for task-specific grasping using deep learning," in *IEEE-RAS International Conference on Humanoid Robots*, no. November, 2017, pp. 91–98.

[49] D. Song, C. H. Ek, K. Huebner, and D. Kragic, "Task-Based Robot Grasp Planning Using Probabilistic Inference," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 546–561, 2015.

[50] M. Mansouri, F. Pecora, and P. Schüller, "Combining task and motion planning: Challenges and guidelines," *Frontiers in Robotics and AI*, vol. 8, 2021.

[51] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[52] J. Leitner, A. W. Tow, N. Sünderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool, P. T. Kujala, L. Nicholson, T. Pham, J. Sergeant, L. Wu, F. Zhang, B. Upcroft, and P. Corke, "The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4705–4712.

[53] S. Park, H. Lee, S. Kim, J. Baek, K. Jang, H. C. Kim, M. Kim, and J. Park, "Robotic furniture assembly: task abstraction, motion planning, and control," *Intelligent Service Robotics*, 2022.

[54] A. A. Apolinarska, M. Pacher, H. Li, N. Cote, R. Pastrana, F. Gramazio, and M. Kohler, "Robotic assembly of timber joints using reinforcement learning," *Automation in Construction*, vol. 125, p. 103569, 5 2021.

[55] A. Thoma, A. Adel, M. Helmreich, T. Wehrle, F. Gramazio, and M. Kohler, "Robotic Fabrication of Bespoke Timber Frame Modules," *Robotic Fabrication in Architecture, Art and Design 2018*, pp. 447–458, 2019.

[56] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot asp solving with clingo," *TPLP*, vol. 19, p. 27–82, 2019.

[57] V. Dietrich, D. Chen, K. M. Wurm, G. V. Wichert, and P. Ennen, "Probabilistic multi-sensor fusion based on signed distance functions," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, 2016, pp. 1873–1878.

[58] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, pp. 303–312, 1996.

[59] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[60] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, p. 163–169, Aug. 1987. [Online]. Available: https://doi.org/10.1145/37402.37422

[61] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[62] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2980–2988, 2017.

[63] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[64] A. Dutta and A. Zisserman, "The VIA annotation software for images, audio and video," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3343031.3350535

[65] V. Labs, "V7 labs," https://www.v7labs.com/, 2023, accessed:8th October 2023.

[66] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9912 LNCS, pp. 483–499, 2016.

[67] S. Ramalingam, S. K. Lodha, and P. Sturm, "A generic structure-from-motion framework," *Computer Vision and Image Understanding*, vol. 103, no. 3, pp. 218–228, 2006, special issue on Omnidirectional Vision and Camera Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314206000695

[68] Scikit-Learn, "Gaussian mixture models," https://scikit-learn.org/stable/modules/mixture.html.

[69] D. Chen, V. Dietrich, Z. Liu, and G. von Wichert, "A Probabilistic Framework for Uncertainty-Aware High-Accuracy Precision Grasping of Unknown Objects," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 90, no. 1-2, pp. 19–43, 2018.

[70] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," *Proceedings of the IEEE Visualization Conference*, no. Section 4, pp. 163–170, 2002.

[71] A. Grunnet-Jepsen, J. N. Sweetser, and J. Woodfill, "Best-known-methods for tuning intel® realsense™ d400 depth cameras for best performance," Intel, Tech. Rep., 2019.

[72] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. Hong, "Analysis and Noise Modeling of the Intel RealSense D435 for Mobile Robots," *2019 16th International Conference on Ubiquitous Robots, UR 2019*, pp. 707–711, 2019.

[73] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.

[74] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "Moveit! task constructor for task-level motion planning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 190–196.

[75] S. Chhatpar and M. Branicky, "Search strategies for peg-in-hole assemblies with position uncertainty," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems.*, vol. 3, 2001.

[76] O. Spector and D. D. Castro, "Insertionnet - a scalable solution for insertion," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5509–5516, 2021.

[77] F. Martín, J. G. Clavero, V. Matellán, and F. J. Rodríguez, "Plansys2: A planning system framework for ros2," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2021, p. 9742–9749. [Online]. Available: https://doi.org/10.1109/IROS51168.2021.9636544

[78] H. Dodampegama and M. Sridharan, "Back to the future: Toward a hybrid architecture for ad hoc teamwork," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 1, pp. 3–10, Jun. 2023. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/25070

A General Manipulation Domain

file: robot_domain_coarse.sp

```
#const numSteps = 1.

sorts
#robot = {rob0}.
#thing = {}.
#object = #robot + #thing.
#place = {input_area,intermediate_area,assembly_area}.
#grasp_mode = {dexterous,vacuum}.
#action = putdown(#robot,#thing), move(#robot,#place), pick_up(#robot,#thing),
    change_grasp_mode(#robot,#grasp_mode).
#boolean = {true, false}.
#outcome = {true, false, undet}.
#inertial_fluent = in_hand(#robot, #thing)+ location(#object, #place)+
    current_grasp_mode(#robot, #grasp_mode).
#fluent = #inertial_fluent + #defined_fluent.
#step = 0..numSteps.

predicates
next_to(#place, #place).
holds(#fluent, #boolean, #step).
occurs(#action, #step).
success().
goal(#step).
something_happened(#step).

rules
% next_to works both ways
next_to(P1,P2):- next_to(P2,P1).
% CWA on next_to
-next_to(P1,P2):- not next_to(P1,P2).
% things can only have a single locaiton at a time
holds(location(T,P1), false, I) :- holds(location(T,P2), true, I), P1!=P2.
% robot can only have one grasping mode at a time
holds(current_grasp_mode(R,G1), false, I) :- holds(current_grasp_mode(R,G2), true, I), G1!=G2.

% objects are removed from the robots hand when they are putdown
holds(in_hand(R,T), false, I+1) :- occurs(putdown(R,T), I).
% robot cannot put down an object which it is not holding
-occurs(putdown(R,T), I) :- not holds(in_hand(R,T), true, I).
% moving updates the robots location
holds(location(R,P), true, I+1) :- occurs(move(R,P), I).
% objects in the robot's hand move with it
holds(location(T,P), true, I+1) :- occurs(move(R,P), I), holds(in_hand(R,T), true, I).
% cannot move to the current location
-occurs(move(R,P1), I) :- holds(location(R,P2), true, I), P1=P2.
% can only move to locations next to the current locaiton
-occurs(move(R,P1), I) :- holds(location(R,P2), true, I), not next_to(P1,P2).
% picking objects up adds them to the robot's grasp
holds(in_hand(R,T), true, I+1) :- occurs(pick_up(R,T), I).
% robot must be at objects location to pick it up
-occurs(pick_up(R,T1), I) :- holds(location(T1,P1), true, I),
    holds(location(R,P2), true, I), P1!=P2.
% robot cannot pick up two objects at once
-occurs(pick_up(R,T1), I) :- holds(in_hand(R,T2), true, I), #thing(T2).
% grasp mode change updates current grasp mode
```

```
holds(current_grasp_mode(R,G), true, I+1) :- occurs(change_grasp_mode(R,G), I).
% grasp mode cannot be changed to the current mode
-occurs(change_grasp_mode(R,G1), I) :- holds(current_grasp_mode(R,G1), true, I), G1=G2.
% cannot change grasp mode whilst holding an object
-occurs(change_grasp_mode(R,G), I) :- holds(in_hand(R,T), true, I), #thing(T).

% planning rules
-holds(F, V2, I) :- holds(F, V1, I), V1!=V2.
holds(F, Y, I+1) :- #inertial_fluent(F), holds(F, Y, I), not -holds(F, Y, I+1), I < numSteps.
-occurs(A,I) :- not occurs(A,I).
success :- goal(I), I <= numSteps.
:- not success.
occurs(A, I) | -occurs(A, I) :- not goal(I).
-occurs(A2, I) :- occurs(A1, I), A1 != A2.
something_happened(I) :- occurs(A, I).
:- not goal(I), not something_happened(I).

% no specified goal
goal(I) :- .

% domain set up
next_to(intermediate_area, assembly_area).
next_to(intermediate_area, input_area).
```

file: robot_domain_fine.sp

```
#const numSteps = 1.

sorts
#robot = {rob0}.
#thing = {}.
#object = #robot + #thing.
#grasp_mode = {dexterous,vacuum}.
#place_c = {input_area,intermediate_area,assembly_area}.
#table_locs = {c1,c2,c3,c4,c5,c6}.
#non_placement_location = {above_input,above_intermediate,above_assembly}.
#place_f = #table_locs + #non_placement_location.
#fine_res_sort = #place_f.
#coarse_res_sort = #place_c.
#action = putdown(#robot,#thing) + move_f(#robot,#place_f) + pick_up(#robot,#thing)
    + change_grasp_mode(#robot,#grasp_mode).
#boolean = {true, false}.
#outcome = {true, false, undet}.
#inertial_fluent = in_hand(#robot, #thing)+ loc_c(#object, #place_c)
    + loc_f(#object, #place_f)+ current_grasp_mode(#robot, #grasp_mode).
#step = 0..numSteps.
#fluent = #inertial_fluent.

predicates
next_to_c(#place_c, #place_c).
next_to_f(#place_f, #place_f).
component(#coarse_res_sort, #fine_res_sort).
holds(#fluent, #boolean, #step).
occurs(#action, #step).
success().
goal(#step).
something_happened(#step).

rules
% only place_f can be a component of place_c
```

```
-component(C1,P1):- #place_c(P1), not #place_f(C1).
% next_to relationships in fine level link coarse locations
next_to_f(C1,C2):- next_to_f(C2,C1).
next_to_c(C1,C2):- next_to_c(C2,C1).
-next_to_f(P1,P2):- not next_to_f(P1,P2).
-next_to_c(P1,P2):- not next_to_c(P1,P2).
next_to_c(P1,P2):- next_to_f(C1,C2), component(P1,C1), component(P2,C2).
% location bridge axiom
holds(loc_c(T,P), true, I) :- holds(loc_f(T,C), true, I), component(P,C).
% can only hold one fine location at a time
holds(loc_f(T,P1), false, I) :- holds(loc_f(T,P2), true, I), P1!=P2.
holds(current_grasp_mode(R,G1), false, I) :- holds(current_grasp_mode(R,G2), true, I), G1!=G2.
holds(in_hand(R,T), false, I+1) :- occurs(putdown(R,T), I).
-occurs(putdown(R,T), I) :- not holds(in_hand(R,T), true, I).
% moving now affects fine location fluent
holds(loc_f(R,P), true, I+1) :- occurs(move_f(R,P), I).
holds(loc_f(T,P), true, I+1) :- occurs(move_f(R,P), I), holds(in_hand(R,T), true, I).
-occurs(move_f(R,P1), I) :- holds(loc_f(R,P2), true, I), P1=P2.
-occurs(move_f(R,P1), I) :- holds(loc_f(R,P2), true, I), not next_to_f(P1,P2).
holds(in_hand(R,T), true, I+1) :- occurs(pick_up(R,T), I).
-occurs(pick_up(R,T1), I) :- holds(in_hand(R,T2), true, I), #thing(T2).
-occurs(pick_up(R,T1), I) :- holds(loc_f(T1,P1), true, I), holds(loc_f(T2,P2), true, I), P1!=P2.
holds(current_grasp_mode(R,G), true, I+1) :- occurs(change_grasp_mode(R,G), I).
-occurs(change_grasp_mode(R,G1), I) :- holds(current_grasp_mode(R,G1), true, I), G1=G2.
-occurs(change_grasp_mode(R,G), I) :- holds(in_hand(R,T), true, I).

% planning rules
-holds(F, V2, I) :- holds(F, V1, I), V1!=V2.
holds(F, Y, I+1) :- #inertial_fluent(F), holds(F, Y, I), not -holds(F, Y, I+1), I < numSteps.
-occurs(A,I) :- not occurs(A,I).
success :- goal(I), I <= numSteps.
:- not success.
occurs(A, I) | -occurs(A, I) :- not goal(I).
-occurs(A2, I) :- occurs(A1, I), A1 != A2.
something_happened(I) :- occurs(A, I).
:- not goal(I), not something_happened(I).

% no specified goal
goal(I) :- .

% domain set up links locations and defines components
next_to_f(above_intermediate_area, above_assembly_area).
next_to_f(above_intermediate_area, above_input_area).
component(c1, intermediate_area).
next_to_f(c1, above_intermediate_area).
component(c2, intermediate_area).
next_to_f(c2, above_intermediate_area).
component(c3, intermediate_area).
next_to_f(c3, above_intermediate_area).
component(c4, intermediate_area).
next_to_f(c4, above_intermediate_area).
component(c5, intermediate_area).
next_to_f(c5, above_intermediate_area).
component(c6, intermediate_area).
next_to_f(c6, above_intermediate_area).
```

B Abstract Beam Domain

file: abstract_beam_domain_D2.sp

```
#const numSteps = 4.
#const startStep = 0.

sorts
#beam = {b7,b4,b5,b8,b3}.
#action = assemble(#beam).
#boolean = {true, false}.
#outcome = {true, false, undet}.
#inertial_fluent = in_assembly_c(#beam)+ supported_c(#beam).
#step = startStep..numSteps.
#fluent = #inertial_fluent.

predicates
fits_into_c(#beam, #beam).
fits_through_c(#beam, #beam).
is_capped_by(#beam, #beam, #beam).
base(#beam).
holds(#fluent, #boolean, #step).
occurs(#action, #step).
success().
goal(#step).
something_happened(#step).

rules
% a beam is capped by two other beams if it fits into both of those beams
is_capped_by(B1,B2,B3):- fits_into_c(B1,B2), fits_into_c(B1,B3), B2!=B3.
is_capped_by(B1,B2,B3):- is_capped_by(B1,B3,B2).
% CWA on capping
-is_capped_by(B1,B2,B3):- not is_capped_by(B1,B2,B3).
% beams are supported if a beam they fit into is already in the assembly
holds(supported_c(B1), true, I) :- holds(in_assembly_c(B2), true, I), fits_into_c(B1,B2).
holds(supported_c(B1), true, I) :- holds(in_assembly_c(B2), true, I), fits_into_c(B2,B1).
% beams cant connect to themselves
-fits_into_c(B1,B2):- B1=B2.
-fits_through_c(B1,B2):- B1=B2.
% cannot have multiple relationships between beams
-fits_through_c(B1,B2):- fits_into_c(B1,B2).
-fits_into_c(B1,B2):- fits_through_c(B1,B2).
% fits are one way relationships
-fits_into_c(B1,B2):- fits_into_c(B2,B1).
-fits_through_c(B1,B2):- fits_through_c(B2,B1).
% CWA on fit relationships
-fits_into_c(B1,B2):- not fits_into_c(B1,B2).

% causal law on assemble action puts beams into the assembly
holds(in_assembly_c(B), true, I+1) :- occurs(assemble(B), I).
% cannot assemble a beam if beams which would cap its ends are already
% in the assembly
-occurs(assemble(B1), I) :- holds(in_assembly_c(B2), true, I),
    holds(in_assembly_c(B3), true, I), is_capped_by(B1,B2,B3), B2!=B3.
% cannot assemble a beam if a beam which needs to be passed through it is already assembled
-occurs(assemble(B1), I) :- holds(in_assembly_c(B2), true, I), fits_through_c(B2,B1).
% cannot assemble a capping beam untill all of the beams it caps are in the assembly
-occurs(assemble(B1), I) :- not holds(in_assembly_c(B2), true, I),
```

```
    holds(in_assembly_c(B3), true, I), is_capped_by(B2,B1,B3), B2!=B3.
% cannot assemble a beam already in the assembly
-occurs(assemble(B), I) :- holds(in_assembly_c(B), true, I).
% cannot assemble beams that are not yet supported
-occurs(assemble(B), I) :- not holds(supported_c(B), true, I).

% planning rules
-holds(F, V2, I) :- holds(F, V1, I), V1!=V2.
holds(F, Y, I+1) :- #inertial_fluent(F), holds(F, Y, I), not -holds(F, Y, I+1), I < numSteps.
-occurs(A,I) :- not occurs(A,I).
success :- goal(I), I <= numSteps.
:- not success.
occurs(A, I) | -occurs(A, I) :- not goal(I).
-occurs(A2, I) :- occurs(A1, I), A1 != A2.
something_happened(I) :- occurs(A, I).
:- not goal(I), not something_happened(I).

% goal definition
goal(I) :- holds(in_assembly_c(b4), true, I) , holds(in_assembly_c(b5), true, I) ,
    holds(in_assembly_c(b8), true, I) , holds(in_assembly_c(b3), true, I).

% domain setup
holds(in_assembly_c(b7),true,0).
fits_into_c(b4,b7).
fits_into_c(b5,b7).
fits_into_c(b4,b8).
fits_into_c(b5,b8).
fits_into_c(b3,b7).
fits_into_c(b3,b8).
base(b7).
```