Improved Robotic Platform to perform Maintenance and Upgrading Roadworks: The HERON Approach

Grant Agreement Number: 955356

D4.3: Representation for High Level Planning (version 1)

Work package	WP4: Motion and High-Level Planner for the automated HERON system
Activity	Task 4.4 “State abstractions for high-level planning” and Task 4.5 “Symbolic action representations for high-level planning”
Deliverable	D4.3: Representation for High Level Planning (version 1)
Authors	Julian Förster, Helen Oleynikova, Olov Andersson, Roland Siegwart
Status	Final (F)
Version	1.0
Dissemination Level	Public (PU)
Document date	30/08/2023
Delivery due date	31/08/2023
Actual delivery date	31/08/2023
Internal Reviewers	Nikolaos Bakalos (ICCS), Eftychios Protopapadakis (ICCS)
	This project has received funding from the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement no 955356.

Document Control Sheet

Version history table			
Version	**Date**	**Modification reason**	**Modifier**
0.1	18/08/2023	Initial draft for internal review	Julian Förster, Helen Oleynikova
0.7	28/08/2023	Draft version after internal quality review	Nikolaos Bakalos, Eftychios Protopapadakis
1.0	2/8/2023	Final version	Julian Förster, Helen Oleynikova, Olov Andersson

Legal Disclaimer

Table of Contents

Contents

List of Tables

List of Figures

Abbreviation Lists

Table 1: Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
AR	Augmented Reality
ASL	Autonomous Systems Lab
BCE	Binary cross-entropy
DOF	Degree of freedom
EC	European Commission
GAN	Generative adversarial network
GNN	Graph neural network
KPI	Key Performance Indicator
MCTS	Monte Carlo Tree Search
MLP	Multi-layer perceptron
OS	Operating System
OT	Optimal Transport
PDDL	Problem domain definition language
PF	Potential Field
RI	Road Intervention
RL	Reinforcement Learning
RoI	Region of Interest
RMP	Riemannian Motion Policy
ROS	Robot Operating System
RUP	Removable Urban Pavement
UGV	Unmanned ground vehicle

Table 2: Abbreviations of the Partners' names

Short name	Participant organization name
ICCS	Institute of Communications and Computer Systems
ACCI	Acciona Construcción S.A.
OLO	Olympia Odos Operation S.A.
UGE	Université Gustave Eiffel
ETHZ	Eidgenössische Technische Hochschule Zürich
ROB	Robotnik Automation
CORTE	Confederation of Organisations in Road Transport Enforcement
STWS	SATWAYS - Oloklиromenes Lyseis Asfaleias Kai Amynas-idiotiki Epicheirisi Parochis Ypiresion Asfaleias (Iepya)-etaireia Periorismenis Efthynis
RISA	RisaSicherheitsanalysen Gmbh
INAC	InnovActs
IKH	Ainoouchaou Pliroforiki SA -IKnowHow-
RG	Resilience Guard Gmbh

1. Executive Summary

This is the third deliverable in WP4 — Motion and High-Level Planner for the automated HERON system – of the HERON project under Grant Agreement No. 955356. Deliverable 4.3 investigates the high-level state and action abstractions required to automate road repair with a robotic arm. This deliverable is a compilation of the work that was completed in the framework of Task 4.4 “State abstractions for high-level planning” and Task 4.5 “Symbolic action representations for high-level planning”.

First, a general overview is presented of the inspection and maintenance processes as envisioned in HERON. From these, the required manipulation capabilities are identified, and corresponding state and action abstractions are derived. In addition to explicitly defining abstractions to be used directly by a symbolic planner for high-level planning, novel solutions are presented for learning abstract states from demonstrations. Additionally, we present a framework for building abstract action models for new and planning goals, that can be used to flexibly and automatically adapt the models that the high-level planners operate on.

The document also contains a brief discussion of the path towards integration of the manipulation skills on the Heron robotic platform in the WP7 integration phase of the project. While some design decisions for the robotic platform are still pending in the project (D5.3, c.f. D4.1), effort was put into covering the expected range of needed manipulation planning capabilities. We provide example software for the manipulation skills but expect that some adaptation will be required later in the project when the robot design is finalized.

2. Introduction

2.1. Purpose of the Document

The document contains deliverable D4.3 “Representation for high level planning”. This is the third deliverable within WP4, the work package on “Motion and High-Level Planner for the automated HERON system” of the HERON project. This deliverable is a compilation of the work that was completed in the framework of Task 4.4 “State abstractions for high-level planning” and Task 4.5 “Symbolic action representations for high-level planning”.

The objective of this deliverable is to research and develop the planning capabilities needed for road maintenance and upgrading tasks, using a robot arm, such as sealing potholes and cracks, painting road markings, as well as removal and placement of road cones and removable urban pavements. Such tasks are challenging from a planning perspective as they are carried out in open-ended environments where the agent can encounter unforeseen situations and disturbances. High-level planning is required to plan the sequence of actions required to achieve a goal, especially in such open environments.

The remainder of this document is organized as follows: first, Section 3 presents an overview of the road intervention tasks in HERON. Then, Section 4 introduces the framework of symbolic planning that we build upon and puts it in the context of needed planning capabilities. Section 5 discusses the challenges of planning in dynamic environments. Sections 6 and 7 introduce the methodology of learning models for symbolic states, and initial symbolic states for HERON. Sections 8 and 9 do the same for actions. Finally, Section 10 finishes with some concluding remarks on the path towards integration of the Heron robot in WP7.

Figure 1: Examples of road infrastructure intervention tasks.

2.2. Intended Audience

Deliverable D4.2 has been designated as public. The information gathered in the document is of special importance to technical partners but can also be of relevance to involved end-users that have been contributing to the scenario definition. The components developed in tasks T4.4 and T4.5 are, as in the rest of WP4, being fed by and are feeding back information to WP2 for the overall architecture of the HERON system.

2.3. Relation with other Deliverables

The outcomes of HERON deliverables D2.1 and D2.2, namely “End-user needs and KPIs report” and “Architecture specification” respectively, serve as the guiding principles while composing the specific document. In particular, D2.1, which is related to the users’ requirements, contains the analysis of current practices, needs, and expectations from infrastructure stakeholders. In parallel, D2.2, which is related to the system architecture and design, includes the specifications of the HERON platform architecture, guidelines, and toolset for development activities. Therefore, the two deliverables directly involve the challenges and

limitations that the AI monitoring framework of the HERON system should analyse and overcome, to efficiently facilitate the RI maintenance procedures.

Deliverable 4.3 presents the high-level planning framework which interacts with all the rest technical and development WPs, such as the perception systems (WP3), the design and construction of the automated UGV system (WP5), and the back-end system that will support the decisions from the road operators and managers (WP6). Subsequently, all these outcomes combined will feed the activities of WP7 field integration for the assessment and validation of the HERON solutions in all the demonstration sites. The initial action representations for Deliverable 4.3, discussed in this document, provide an initial starting point for the integration of all of the partners' work into a single cohesive system, and will be further built on in Deliverable 4.4 with the actual implementation of the high-level planner.

3. Manipulation Challenges in Road Infrastructure Intervention

Road maintenance is a vital task for ensuring safe and efficient roadways, which are the backbone of our interconnected society. Despite the shift to more digitalization and automation in every industry, road maintenance and construction remains an area where automation has so far not been fully exploited.

To this end, the Heron project aims to increase the productivity of road intervention procedures by designing a (semi-) automated system with a multi-degree-of-freedom (MDOF) robotized vehicle capable of aiding in multiple road maintenance tasks. By leveraging recent advances in artificial intelligence and machine learning techniques combined with state-of-the-art robotics sensors and actuators (see Figure 2), perception, manipulation and planning algorithms to perform maintenance and upgrading of roadworks. By using advanced data coming from various sources (including V2I and aerial drone surveillance) and well-established methods (from existing know-how from research and industrial projects), the automated system will be able to provide a level of non-routine (emergency) maintenance operations when required.

HERON targets the development and prototype validation of an innovative, automated intelligent robotic platform able to perform:

- Cone placing and picking
- Potholes repairing

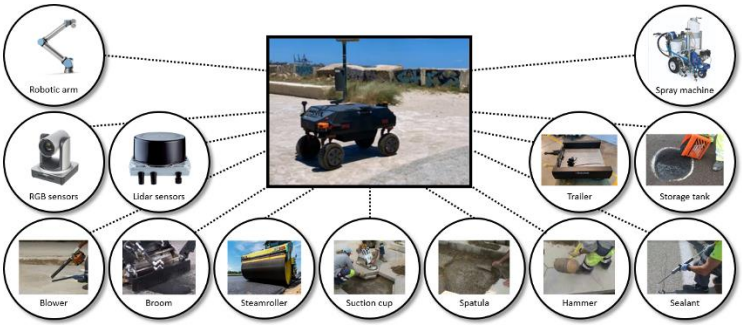


Figure 2: From traditional tools to robotic sensors and actuators (final design pending on D5.3).

Figure 3: Placing traffic cones.

- Cracks repairing
- Road marks painting
- RUP (Removable Urban Pavement) placing

As opposed to factory applications, where robots operate in clean, closed-off, and highly controlled environments, road maintenance takes place in dirty and changing outdoor environments. In addition to the harsh outdoor environments, road maintenance requires the skilful combination of a wide variety of complex manipulation skills and the ability to adapt these skills to different embodiments of road defects (every pothole looks different).

In this deliverable, we investigate the novel planning capabilities required for automating road maintenance tasks with robots. To this end, the Heron project will develop a robotic system whose capabilities will be showcased with the five different and common road maintenance tasks above, each with its unique challenges.

We will now describe the considered RI intervention tasks and highlight their challenges in terms of robot high-level planning. This will lay the foundation for discussing the requirements for the Heron high-level planning solution, which will be discussed in the following chapters.

3.1. Traffic Cone Placement and Removal

Every road intervention procedure starts and ends with blocking off the road segment to be maintained and removing it from traffic. Traffic cones (Figure 3) are the most important tools in this separation process.

The Heron robotic system needs to pick up the traffic cones from the storage area on the platform and place them at the correct target location. This will require the planner to identify suitable and collision free cone target poses, and the determine the right sequence and parameterization of navigation, grasping and placing actions. Replanning upon encountering unforeseen obstacles and situations is required.

3.2. Filling Road Cracks

Road cracks (Figure 4) are a frequent road damage that can increase the risk of accidents and accelerate the deterioration of the road substance and lead to larger damages, such as potholes.

Fixing road cracks requires the robotic system to use its arm to deposit the correct amount of sealant material along the crack, which can have varied shape and size, and potentially require contact-rich interaction to precisely distribute even out the deposited sealant material, typically done with a squeegee (D2.1). If needed, this could be mounted on the arm as well.

Figure 4: Filling road cracks.

Figure 5: Faded road markings.

3.3. Renewing Road Markings

Clearly visible road markings are required for safely and efficiently guiding the traffic flow.

However, due to deterioration caused by their exposure to traffic and weather conditions (see Figure 5), they need to be regularly renewed. Renewing road markings will require the robotic system to precisely spray paint where the road marking's quality has dropped below a minimum threshold and do this while avoiding spillage of paint. In terms of requirements on high-level planning, this is similar to crack sealing in that a linear segment needs to be tracked by the end-effector, but no contact with the ground is required.

3.4. Patching Potholes

Patching potholes is a more challenging task, compared to the sealing crack process; the material just needs to be deposited to cover a 2D pothole surface instead of along the curve of a crack.

While there are many ways to patch a pothole, the correct amount of filling material needs to be deposited, and depending on the material, distributed across the pothole, and then levelled and compacted by the robot until a smooth target state of the material is achieved (see Figure 6). Involving a number of tools and steps, which depend on one another's success, requires a carefully designed high-level planner.

(a) Depositing filling material (b) Flattening surface

Figure 6: Two steps in the process of fixing a pothole.

Figure 7: Placing of RUPs.

3.5. Inserting and Removing Road Tiles (RUPs/CUDs)

Removable Urban Pavements (RUPs) (from French: Chaussée Urbaine Démontable, short CUD, see Figure 7) [1] are an innovative solution for modular road construction. RUPs replace monolithic road segments, poured from asphalt, with a set of concrete slabs that can be individually placed and removed. This modular system enables a cleaner and more localized method to fix road defects without affecting the entire road, as well as easy access to infrastructure that might lie below the road surface, in case of required repair works. The slabs come in two variants, with the newer one interlocking (shown in the image), and one with independent tiles.

Placing and removing RUPs will require the robot system to handle heavy and inert concrete slabs that require accurate motions for precisely placing the tiles at their target location under contact with the other tiles and the ground. In addition, preparing the ground for placement of RUPs will require levelling of the underlying surface material (e.g., sand).

4. Symbolic planning, state and action abstractions and their groundings in the context of HERON

Robotic agents navigating in, and interacting with, complex environments face challenging planning problems. Taking the robot itself as well as all entities in its surroundings into account, a high dimensional configuration space results. Even slight changes in the robot's configuration during an interaction can lead to vastly different state space modes and thus outcomes of the interaction. To deal with this complexity, a common approach is to introduce a level of abstraction, lifting a planning problem to a lower-dimensional and abstracted, and thus tractable one.

A dominant planning methodology leveraging abstractions is symbolic planning [2]. It operates based on state abstractions that are also referred to as predicates. A predicate is a function of the state of the world which returns a binary value, signalling whether the predicate holds (true) or not (false). In addition, a predicate can take one or multiple arguments, allowing it to model a general concept that can be evaluated for different entities. An example for a predicate is "in hand", taking one argument. It evaluates to true if the entity passed as argument to the predicate is in the robot's hand, and to false otherwise.

In order to allow initialising a planning problem from the current state of the robot's environment, or to execute devised plans using the robot, a connection needs to be made between the abstract planning model and the real world. This is referred to as grounding the symbolic model. For abstract states, a grounding is used to translate between real world states and symbolic states. Given a real-world state, classifiers can be used to determine a symbolic state based on perception information, constituting a translation to a symbolic state. Vice versa, generators are required to sample states in the real world, such that given abstract states are fulfilled. The sampled states can then be used to parameterize robot actions.

Like states, actions need to be abstracted as well to be used by a symbolic planner. Action abstractions model preconditions that need to hold before the action can be executed (e.g., the traffic cone needs to be in the robot's hand before it can be placed at a desired location), as well as effects. Effects model state changes caused by executing the action. Both preconditions and effects in an action abstraction are expressed in terms of predicates.

Many of the manipulation challenges for the HERON project, introduced in Section 2, require a long sequence of steps to solve. The high-level planning system does not need to decide the order of actions to carry out, but also action parameterizations. Often executing one action depends on the success of a previous ones. In case of unexpected obstacles or failure of an action in the sequence, corrective actions need to be taken. Symbolic planning as high-level planning framework is very conducive to these requirements. Both actions and states can be compactly modelled, and planning times are very low which enables replanning in case of failure to achieve action goals. Furthermore, modelling action outcomes explicitly and introducing classifiers to test them based on perception data is paramount for introducing reactivity and resilience to the system.

5. Symbolic planning in dynamic environments

If planning is required for a set of tasks that is completely specified at design time, working with a predefined set of symbolic states and actions is sufficient. However, once either tasks and/or environments change, we want to apply robotic systems to become more open-ended, a fixed set of symbolic states and actions will keep the robotic agent from achieving new and unseen goals. Therefore, the abstract planning model needs to be extended upon encountering new tasks or situations. Doing this manually requires an expert, which renders the adaptation expensive and unscalable. Instead, our vision is that the grounding for new symbolic states can be learned from labelled examples, resulting for example from demonstrations given by non-expert users. Similarly, action abstractions should be extracted from data collected during a robot's interactions with an environment. In the following, we present methods we developed towards achieving this vision. Specifically, we introduce an architecture and a training procedure for learning the two components necessary to ground a symbolic state: a classifier and a generator. This is presented in Section 6. For simplifying the introduction of new action abstractions, we developed an algorithm to learn state abstractions from interaction data, which is detailed in Section 8.

6. Learning abstract state groundings from demonstrations

6.1. Method

Overview

Grounding a predicate for the purpose of planning requires two components. First, a classifier needs to decide if the predicate holds or not, given an observation of the environment. Second, a generator is needed to output a configuration of entities in the environment such that the given predicate holds. This target configuration can then be used in the planning process to parameterize an action to achieve the desired state.

We employ learning-based models as classifiers and generators to allow training on demonstrations that were given by non-expert users of the robot. This enables more flexible robots since an agent's planning model can be extended after deployment to handle new tasks and situations the robot encounters.

Targeted predicates

For this work we focus on predicates that model geometric relations between rigid objects. The main reason for this is that while such predicates can be modelled based on geometric information, they cover a very large space of predicates that is not only task but also preference dependent. Therefore, learning models for such predicates instead of defining all models for predicates a robot might encounter at design time is very desirable.

Features

Since predicates model general concepts, they can take as arguments one or multiple entities from the scene that the concept is to be applied to. Features need to be selected to serve as inputs to the learning-based models. The first set of features, referred to as *manual features*, consists of:

- Centroid position
- Orientation
- Axis-aligned bounding
- Object-aligned bounding box

This information can be extracted using a scene segmentation framework. The difference between axis- and object-aligned bounding box is the object orientation in which it is computed. While the axis-aligned bounding box captures the extent of the object in its current position and orientation, the object-aligned bounding box is computed with the object in a nominal orientation, often such that the bounding box is aligned with the object's principal axes.

While these features should allow deciding many of the predicates of interest, some predicates depend on finer geometric details. Furthermore, depending on a per object canonical frame still requires some degree of manual engineering. For this reason, we investigated using point cloud encodings as features. Assuming segmented point clouds, PointNet [3] as well as convolutional models were investigated as encoders. Training was conducted through an auto-encoding scheme, where the encoder is a network based on PointNet, convolutional, or fully connected layers. Once the encoder is trained, its weights are frozen for the downstream

predicate learning. Since the point clouds were normalised before being encoded, the final feature vector for an object is composed as:

- Centroid position
- Orientation
- Point cloud encoding.

Data generation

While we envision training predicate models based on user demonstrations, for the purpose of developing the methodology, we generated data using a physics simulation. For this, we focused on two predicates:

"on_clutter"	takes two arguments and indicates whether one of the argument objects is placed on top of the other argument, for example a cup placed on top of a table.
"inside_drawer"	takes two arguments and models whether one of the objects is placed inside the other.

In the physics simulation, we generate different types of positive and negative samples, i.e. samples where the predicates hold and where they do not, respectively. For each sample, features are extracted, either the bounding boxes to form the first type of features described above or point cloud encodings for the second feature type. Based on the resulting datasets of samples, classifier and generator models are trained.

Classifier architecture

The simplest model that can be used as a classifier is a multi-layer perceptron (MLP). Taking as input the concatenated features of the argument objects, it outputs a single value that represents the prediction of whether the predicate holds or not for the given argument objects.

We observed that in practice, not only argument objects can play a role in predicate fulfilment, but also surrounding objects in the scene. For this reason, we also investigated models that can take surrounding objects into account. A challenge here is that this requires the network to deal with varying numbers of inputs, which an MLP for example is not equipped for. Instead, we turned to graph neural networks (GNN). Intuitively, our network computes a scene encoding based on all surrounding object features that were concatenated with the argument object features. Finally, an MLP takes in the argument object features, as well as the scene encoding to produce a predicate prediction. A schematic overview of all models is shown in Figure 8.

Generator architecture

Generators are realised using an architecture similar to the classifier implementation. On one hand, we investigated using MLPs as generators. While the inputs to the generator are identical to the classifier, its output consists of position and orientation the argument objects should have to achieve the predicate the generator was trained for. In a post processing step, both axis-aligned and object-aligned bounding boxes are computed using the ones from the input feature vectors and the newly computed positions and orientations.

Similar to the classifier, we also investigated a GNN based generator architecture. Here, a scene encoding is computed from the features of all surrounding objects that are concatenated with the argument object features for context (see Figure 8). Then, an MLP takes argument object features and scene encoding to predict new positions and orientations for the argument objects.

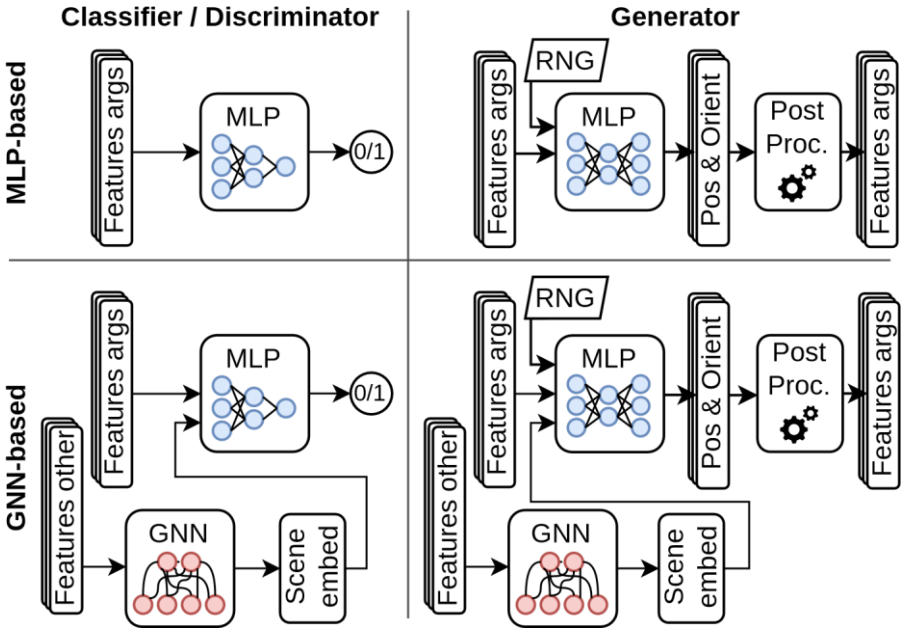


Figure 8: Schematic overview of the models we propose for predicate classification and generation. While MLP models only consider argument objects of a predicate, GNN models can also take surrounding objects into account.

Training procedures

Given data and the model architecture, we need training procedures for classifiers and generators. To keep the models simple and to separate the effect of different predicates, we opted for training a separate classifier and generator for every predicate.

Classifier training was conducted in a supervised fashion, using a standard binary cross-entropy loss and the ADAM optimizer. This one and the training procedures described in the following are visualized in Figure 9.

To train the generator, three different training schemes were explored. First, the generator was supervised using the pre-trained classifier. With the classifier's weight frozen, samples generated by the generator were scored with a loss function that has a high output if the generated sample is classified as "false" (predicate does not hold according to classifier) and a low output if the sample is classified as "true" (predicate does hold).

For the second generator training scheme, a generative adversarial network (GAN) style training was applied [4, 5]. For this, we introduced a third network: a discriminator. Its architecture is identical to the classifier. However, its role is not to determine whether a predicate holds or not, but whether a sample input to it comes from the dataset or was generated. Being trained at the same time together with the generator, this adversarial scheme incentivizes the generator to produce an output distribution that is similar to the distribution found in the training set. For this reason, the discriminator is only fed with positive samples from the training set, to ensure that generated samples are positive.

A third generator training scheme combines the first two, summing the loss from classifying a generated sample and from passing it through the discriminator.

6.2.Evaluation

We conducted a number of experiments to evaluate the performance of our proposed predicate learning framework. Here, we present results for the classifier and the generator.

As a baseline, we trained decision tree classifiers on the same data that our proposed models were trained with.

Classifier

The results for different classifier configurations are shown in Figure 10. Every classifier was trained for 20'000 iterations on the full dataset, each iteration constituting an optimization step after backpropagating over a single mini batch.

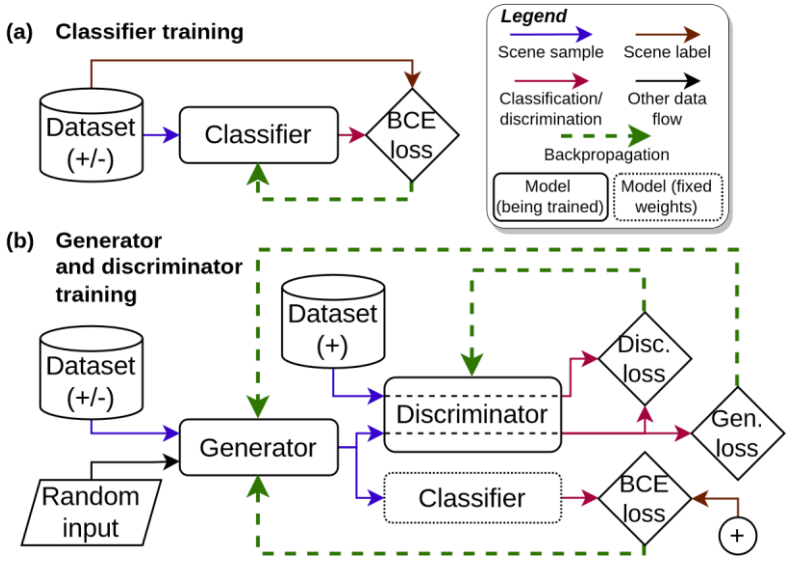
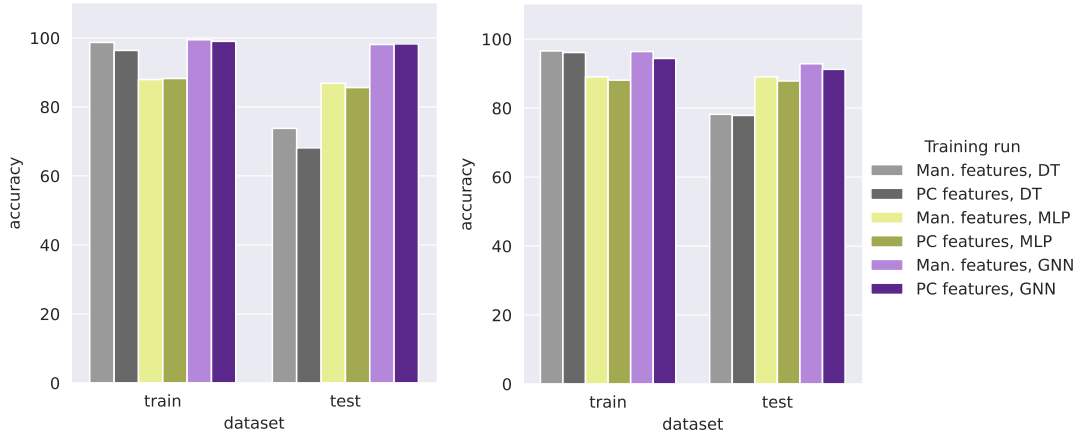


Figure 9: Schematic overview of the training procedures proposed for training predicate classifiers, generators and discriminators.



a) Accuracy for training for the **"on_clutter"** predicate.

b) Accuracy for training for the **"inside_drawer"** predicate.

Figure 10: Accuracy of different classifier configurations.

We observe that the baseline decision tree classifiers achieve a good performance on the training dataset both for the "on clutter" and the "inside drawer" predicate. However, in both cases the test accuracy is reduced dramatically. This indicates overfitting.

Compared to this, the MLP models achieves better generalization. However, compared to our proposed GNN models, the MLPs still fall behind. We attribute this to the fact that the GNN models can take surrounding objects into account, correctly predicting the predicate as not holding when another object is in collision with the object the network is queried about. This is highlighted in Figure 11, where it can be observed that the GNN models correctly

classify negative samples where the object of interest is in collision with another object ("neg-obj on collision").

Furthermore, we observe that the models predicting based on the point cloud embeddings perform close to on par with the models that are based on manual features. Even though point cloud-based models need to learn to extract geometric information from the point cloud embeddings, it is promising to see that this does not significantly heart prediction accuracy.

Generator

For the purpose of evaluating generators, we manually implemented classifiers for judging whether a generated sample fulfils the predicate. The dataset used for training is the same as for the classifier training and thus contains positive and negative sample scene, in which the predicate of interest already holds or does not respectively. Then, for each sample scene from the dataset used as input to the generator, nine random seeds were used to have the generator output nine scene samples for which the predicate should hold. Results obtained from the test dataset (which was excluded from training) are shown in Figure 12.

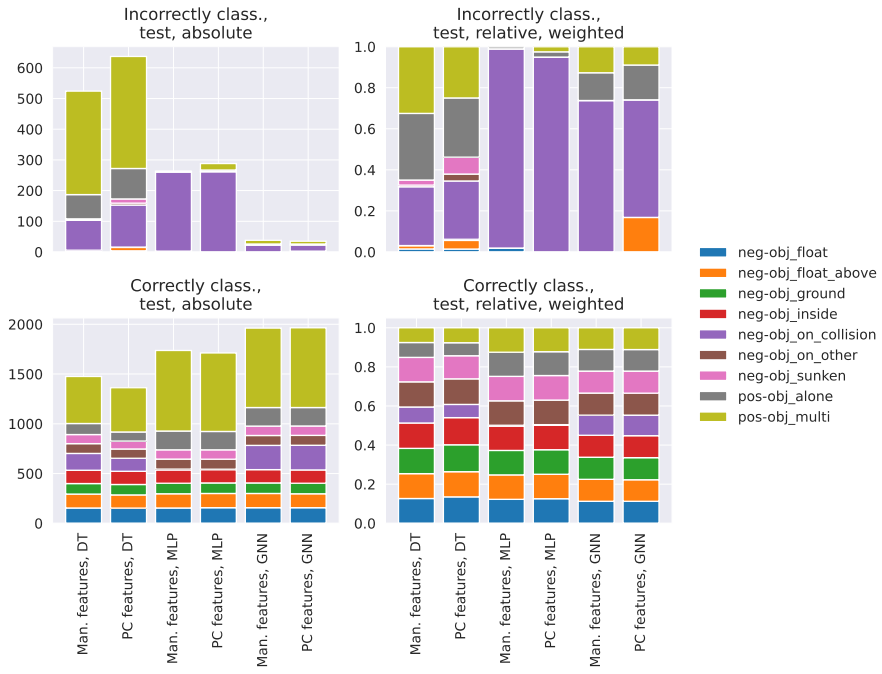
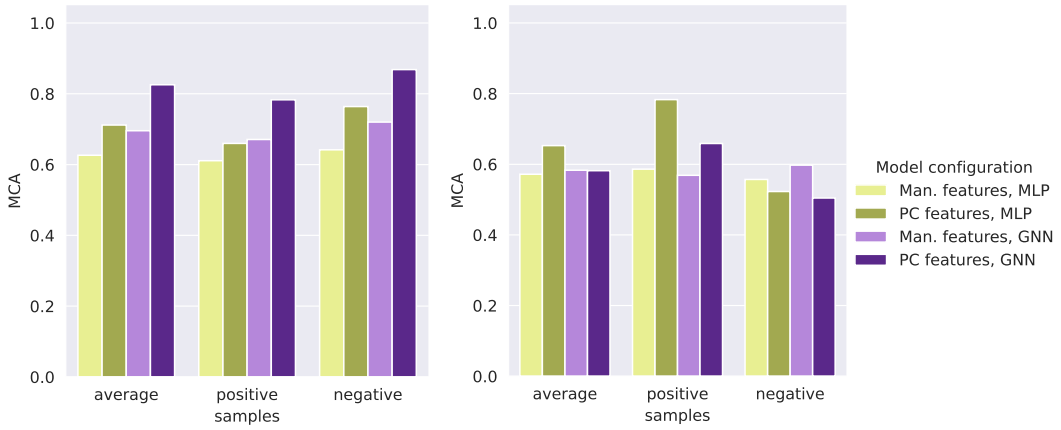


Figure 11: Performance of different classifier configurations for the "on clutter" predicate. The colors within a bar and labelled in the legend represent sample types. This gives an insight into how each classifier configuration fares on a certain sample type.



(a) Success rate for sampling generators for the **"on_clutter"** predicate.

(b) Success rate for sampling generators for the **"inside_drawer"** predicate.

Figure 12: Performance of different generator architectures and feature types, obtained on the test dataset, split up for positive and negative input samples, as well as the average over both.

For the "on clutter" predicate, our best generator configuration achieves a success rate of more than 80 %. This is achieved with point cloud features as input and using the GNN architecture. For the MLP architecture, using point clouds features as input also results in a better performance than when using manual features. However, taking surrounding objects into account is indicated to be important by the results.

For the "inside drawer" predicate instead, our GNN architecture does not profit from the point cloud features and performs worse than the MLP configuration when relying on point cloud features. A possible explanation is that for the "inside drawer" predicate, the surrounding objects that we want to avoid collisions with are less important. Instead, the orientation of objects plays a bigger role to avoid the object to protrude from the drawer's bounding box, which would prevent the drawer from closing.

Overall, a generator success rate of around 60% can strongly benefit planning. The alternatives to targeted sampling using our generators are either manually engineering generators for every new predicate or relying on generic uniform sampling of action parameters. While the former approach does not scale to applications in open-ended environments and non-export users, the latter approach yields very low success rates.

7. Initial state abstractions for HERON

This section introduces the state abstractions (also referred to as predicates) needed to formulate planning problems towards achieving the RI activities outlined in Section 2. The abstractions are amenable to a planning domain definition language (PDDL) formulation of the problem. Apart from the parameters each predicate takes as inputs, we describe how the predicate is intended to be grounded. Grounding refers to the method connecting a predicate to the real world.

The format we use to represent predicates is borrowed from PDDL. First, the predicate's name is given in all capital letters. This is followed by the predicate's arguments, recognizable by a "?" preceding every argument. A predicate can be initialized by replacing the arguments with concrete entities, which would be stated without the "?".

7.1. Traffic Cone Placement and Removal

- AT ?cone ?target

Parameters ?cone: a traffic cone, ?target: a target pose for the cone

Description Returns *true* if the cone is at the target pose, and *false* if not.

Grounding Bounding box information from the segmentation map from the perception pipeline.

- IN_HAND ?cone

Parameters ?cone: a traffic cone

Description Returns *true* if the robot holds the cone in its hand, and *false* if not.

Grounding Proprioceptive data collected from sensors in the robot's hand.

- IN_REACH ?target

Parameters ?target: a grasp or placement pose for the end-effector

Description Returns *true* if the target is in reach of the robot arm such that it can pick or place an object it holds there, and *false* if not.

Grounding Inverse kinematics based on the robot's current position as reported by the localization system.

- CFREE ?cone ?target

Parameters ?cone: a traffic cone, ?target: a target pose for the cone

Description Returns *true* if there is no collision with other objects if the cone was placed at the target, and *false* if there is.

Grounding Bounding box information from the segmentation map from the perception pipeline. • EMPTY HAND

Description Returns *true* if the robot's hand is empty, and *false* if there is.

Grounding Proprioceptive data collected from sensors in the robot's hand.

7.2.Filling Road Cracks

- CLEAN ?crack

Parameters ?crack: a road crack

Description Returns *true* if the crack is clean from debris, and *false* if not.

Grounding Classification based on camera images.

- DEPOSITED ?crack

Parameters ?crack: a road crack

Description Returns *true* if the crack is completely filled with the filling material, and *false* if not.

Grounding Classification based on camera images.

- FLATTENED ?crack

Parameters ?crack: a road crack

Description Returns *true* if the material deposited on the target crack compressed to form a flat surface, and *false* if not.

Grounding Classification based on camera images.

- AT ?crack

Parameters ?crack: a road crack

Description Returns *true* if the robot is positioned at the crack, in reach to operate on it, and *false* if not.

Grounding Based on robot pose reported by localization system and crack position and extent reported by perception pipeline.

7.3.Renewing Road Markings

- FADED ?marking

Parameters ?marking: a segment of a road marking

Description Returns *true* if the marking within the segment is faded and needs repainting, and *false* if not.

Grounding Classification based on camera images.

- CLEAN ?marking

Parameters ?marking: a segment of a road marking

Description Returns *true* if the is clean from debris and dust, and *false* if not.

Grounding Classification based on camera images.

- AT_START ?marking

Parameters ?marking: a segment of a road marking

Description Returns *true* if the robot is positioned at the start of the crack segmented and aligned with it, and *false* if not.

Grounding Based on robot pose reported by localization system and marking position and extent reported by perception pipeline.

7.4.Patching Potholes

- CLEAN ?pothole

Parameters ?pothole: a segment of a pothole

Description Returns *true* if the segment is clean from debris, and *false* if not.

Grounding Classification based on camera images ((depth and RGB).

- DEPOSITED ?pothole

Parameters ?pothole: a segment of a pothole

Description Returns *true* if the pothole segment is completely filled with the filling material, and *false* if not.

Grounding Classification based on camera images (depth and RGB).

- FLATTENED ?pothole

Parameters ?pothole: a segment of a pothole

Description Returns *true* if the material deposited on the target crack compressed to form a flat surface, and *false* if not.

Grounding Classification based on camera images.

- AT ?pothole

Parameters ?pothole: a segment of a pothole

Description Returns *true* if the robot is positioned at the pothole, in reach to operate on it, and *false* if not.

Grounding Based on robot pose reported by localization system and pothole position and extent reported by perception pipeline.

7.5. Inserting and Removing Road Tiles (RUPs/CUDs)

- AT ?tile ?target

Parameters ?tile: a road tile, ?target: a target pose for the road tile

Description Returns *true* if the road tile is at the target pose, and *false* if not.

Grounding Bounding box information from the segmentation map from the perception pipeline.

- IN_HAND ?tile

Parameters ?tile: a road tile

Description Returns *true* if the robot holds the road tile in its hand, and *false* if not.

Grounding Proprioceptive data collected from sensors in the robot's hand.

- IN_REACH ?target

Parameters ?target: a target pose for the road tile

Description Returns *true* if the target is in reach of the robot arm such that it can place an object it holds there, and *false* if not.

Grounding Inverse kinematics based on the robot's current position as reported by the localization system.

- CFREE ?tile ?target

Parameters ?tile: a road tile, ?target: a target pose for the road tile

Description Returns *true* if there is no collision with other objects if the road tile was placed at the target, and *false* if there is.

Grounding Bounding box information from the segmentation map from the perception pipeline.

8. Learning abstract action models from interaction data

8.1. Motivation

Now that we have a way of learning predicate groundings from data instead of manually designing classifiers and generators for new predicates, we still need to simplify the second important component of the planning model used by symbolic planners: actions. Therefore, instead of manually modelling all actions, we present an algorithm to extract action descriptions from goal-oriented robot interaction data.

For this, we assume that a robotic agent is equipped with a basic set of skills. These skills can be sequenced to achieve a large number of outcomes. Outcomes that can be achieved depend on the agent itself, the environments, the specific objects involved in an interaction, the task to be achieved, preferences by users, etc. Modelling all this at design time is infeasible, so even though the robot could in principle achieve all these outcomes with the set of basic skills it is equipped with, the bottleneck lies with the planner. Due to the large space of potential outcomes, it is not possible to make the symbolic model expressive enough so that the planner can plan towards all of these outcomes.

To alleviate this limitation, we developed an algorithm that explores sequences of the basic skills the robot is equipped with. The algorithm is intended to be applied in two scenarios. First, if a new user-defined goal is to be achieved by the robot, we assume that the goal can be achieved with the set of basic skills the robot is equipped with and attempt to find a successful sequence. Second, if the robot encounters an unmodelled obstacle and thus sequences output by the task planner fail, a new sequence taking the obstacle into account needs to be discovered. Upon finding a successful sequence in either case, our algorithm extends the symbolic description appropriately, to allow the task planner to plan towards achieving this and similar goals in the future.

The work described in this section was published in the Robotics and Autonomous Systems journal for wider dissemination to the robotics research community. For more details, please refer to the publication [6].

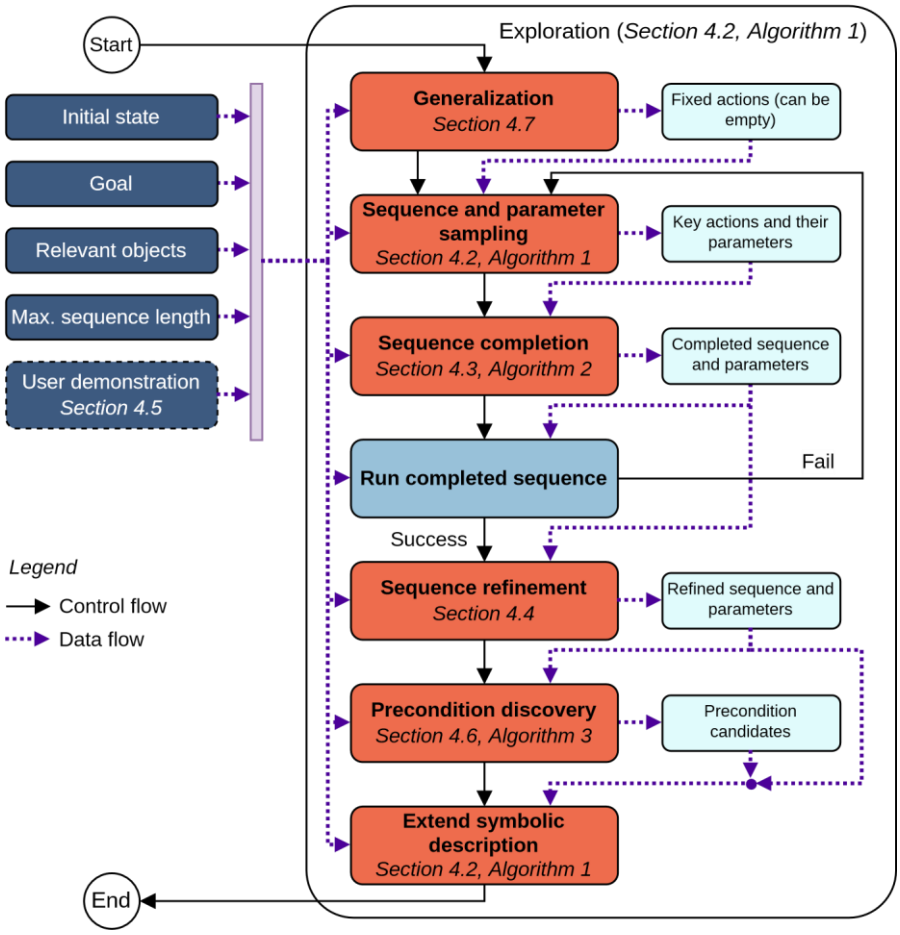


Figure 13: Overview of the proposed skill set exploration algorithm.

8.2.Method

The proposed algorithm is shown schematically in Figure 13. In its core, it samples sequences ("Sequence and parameter sampling" in Figure 13), tests them for success ("Run completed sequence"). If not successful, it continues sampling sequences. If successful, the symbolic description is extended as needed ("Extend symbolic description"). All other modules highlighted in the figure are to make the exploration more efficient.

Our *sequence completion* module allows to reduce the search space over sequences significantly. This is achieved by using the symbolic planner to complete sampled sequences into longer feasible ones. This is possible since every action has preconditions which can be used as a goal for a planning problem. The planner will then output a sequence to fulfill the preconditions. Collecting the filling actions for every action in the originally sampled sequence yields a much longer, but feasible sequence, thus allowing as to sample over the smaller space of shorter sequences while still being able to find sequences successfully solving longer horizon problems.

Since the length of the successful sequence is unknown at exploration time, we opt for sampling sequences of a long upper bound length. Then, often, the steps to successfully achieve the goal are only a subset of the actions in the whole sequence. To reduce to the minimum number of steps sequence that still achieves the goal, we implemented a *sequence refinement* module. It iteratively and systematically drops parts of the sequence, completes the remainder using the sequence completion component and tests it for success. The shortest successful sequence is passed on.

Next, we perform *precondition discovery*. The goal here is to split the successful sequence into atomic components that are linked through dependency-injecting predicates. This allows the planner to sequence only the parts of the sequence as needed to set the preconditions for the action(s) that achieve(s) the goal, and to finally achieve the goal. If a part of the sequence is not needed because the corresponding precondition is already met, it is skipped, which would not be possible without the precondition discovery step.

When starting the exploration algorithm, we aim to use experience collected during previous runs of the algorithm to speed up the search. For this, the *generalization* module tests using the symbolic planner whether any action can be used to achieve the new goal if the new objects are allowed to be parameters for any action in the set of symbolic actions. If this is successful, the symbolic description is extended to permanently allow the new entities as parameters to the relevant actions, thus dramatically reducing the search effort.

In a similar attempt to reduce the search effort, our algorithm is capable of taking user demonstrations into account. A demonstration consists of actions that are part of a successful sequence, along with parameters for these actions. There are no requirements on the completeness of a demonstration. Any parts of it that are not provided will be explored by our algorithm in the fashion described above. While being optional, a user demonstration can be simple for a non-export user to provide, while significantly narrowing the search space needed to cover by our algorithm.

8.3.Evaluation

We conducted an extensive series of experiments to test our algorithm. As a baseline, we opted for a Monte Carlo Tree Search (MCTS) algorithm [7] that has access to the same set of basic actions and progressively builds a tree of action sequences until a successful sequence is discovered.

We chose a rearrangement task domain for the experiments. Here, a mobile manipulation platform is equipped with the following four basic skills:

Navigate the robot to a desired pose,

Grasp an object in reach,

Place an object that was previously grasped at a desired location, and

Move an articulated mechanism such as a drawer of door.

Using these skills, the robot was tasked with achieving the goals outlined in Table 1. For every scenario, there is a time budget of 1000 seconds available for our algorithm or the baseline to find a solution to the problem. Upon depleting budget, a run is considered a failure. The tasks are of increasing length and complexity, requiring more and more steps to achieve as outlined in the table. Furthermore, in some scenarios, the agent is initialized with experience gained from a previous execution of the exploration algorithm. This is to test the generalization capabilities of our algorithm. An impression of the domain is given in Figure 14.

Apart from our algorithm and the baseline, we also tested our algorithm when supplied with user demonstrations. The demonstrations given for each scenario are listed in Table 2.

Numeric results from running the experiments are shown in Figure 15. We observe that our method outperforms the baseline in all scenarios in terms of run time, and in all but one scenario in terms of success rate. We attribute this to sequence completion which reduces the search space that need to be covered.

Table 1: Experiment scenarios in the rearrangement task domain. "ID" stands for experiment ID, l is the minimum length of a successful sequence, l_k refers to the minimum number of key actions in a successful sequence, and "Prior" refers to the information available to the algorithm at planning time, consisting of the agent's experience after the specified scenario ID.

ID	Goal	Initial state	l	l_k	Prior
(a)	Cube on cupboard	Cube lying on table	4	1	None
(b)	Tall box inside shelf	Tall box standing on table	4	1	None
(c1)	Cube inside container1	Cube on table, container1 covered with lid	8	2	None
(c2)	Duck inside container1	Duck on table, container1 covered with lid	8	2	After (c1)
(c3)	Cube inside container2	Cube on table, container2 covered with lid	8	2	After (c1)
(c4)	Duck inside container2	Duck on table, container2 covered with lid	8	2	After (c1)
(d1)	Cube inside container2	Cube inside container1, both containers covered with lids	12	3	None
(d2)	Duck inside container2	Duck inside container1, both containers covered with lids	12	3	After (d1)

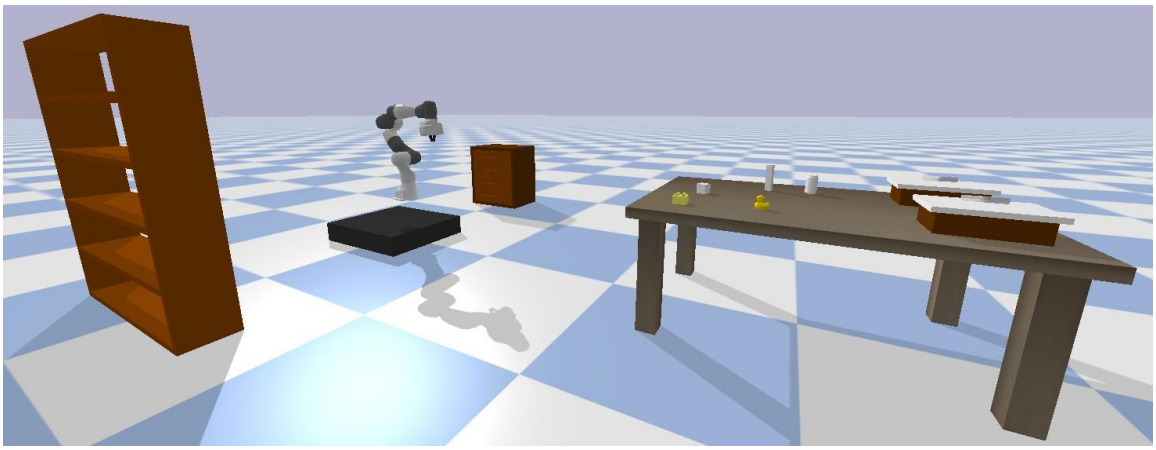


Figure 14: The rearrangement task domain we evaluated our skill set exploration algorithm in.

Table 2: Demonstrations supplied to our method for the experiment IDs defined in Table 1.

ID	Sequence Demo	Parameter Demo
(a)	["place"]	[{"obj": "cube1"}]
(b)	["place"]	[{"obj": "tall box"}]
(c1)	["place", "place"]	[{"obj": "lid1"}, {"obj": "cube1"}]
(c2)	["place", "place"]	[{"obj": "lid1"}, {"obj": "duck"}]
(c3)	["place", "place"]	[{"obj": "lid2"}, {"obj": "cube1"}]
(c4)	["place", "place"]	[{"obj": "lid2"}, {"obj": "duck"}]
(d1)	["place", "place", "place"]	[{"obj": "lid1"}, {"obj": "lid2"}, {"obj": "cube2"}]
(d2)	["place", "place", "place"]	[{"obj": "lid1"}, {"obj": "lid2"}, {"obj": "duck"}]

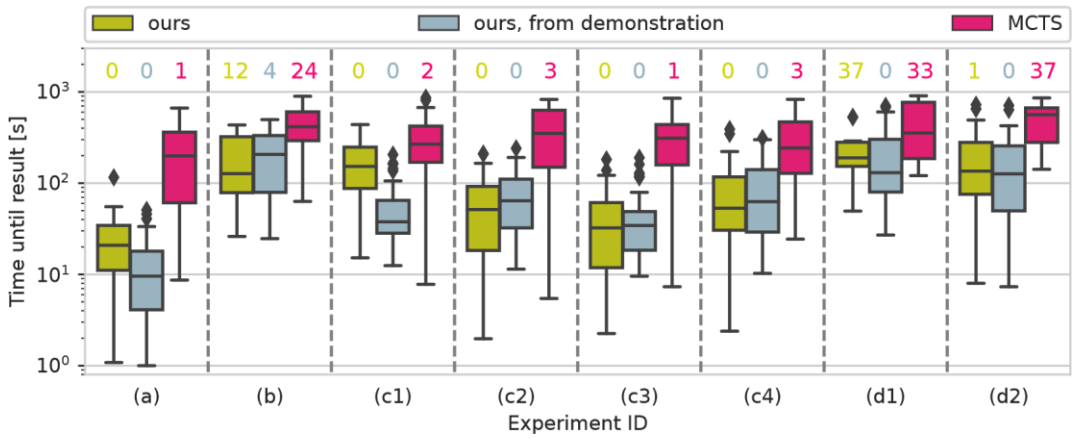


Figure 15: Numeric results for evaluating our method and the baseline on the experiment scenarios shown in Table 1. Every method was ran 50 times on each scenario. The numbers in the top are failures over the 50 runs (lower is better).

Additionally, for runs (c2)-(c4), our algorithm is initialized with the previous knowledge gained after (c1). The generalization component helps reducing the exploration time needed here for subsequent runs. This is even more pronounced in scenario (d2), which is initialized with experience from scenario (d1). There, the success rate can be improved from 26 % in (d1) to 98 % in (d2) thanks to generalization, while the MCTS baseline achieves success rates of 34 % and 26 % respectively.

The results also highlight the utility of user demonstrations. In all runs but (b), the success rate lies at 100 %. Run (b) requires sampling a placement pose with a high accuracy, which is not part of the demonstration. Since we rely on uniform parameter sampling in this evaluation, the probability of sampling a successful placement pose is low. This could be alleviated by employing the framework presented in Section 5 as targeted sampler.

In Figure 16, we present qualitative results from applying our algorithm to experiment scenario (c1), highlighting the role of precondition discovery. In Figure 16a, we have an action to place an object into a container (see Figure 14). Part of the successful sequence was to remove the lid from the container before the cube can be placed inside. However, since removing the lid is only necessary when the lid is present on the container in the first place, our precondition discovery component created a separate action for removing the lid. The two

actions are connected using the dependency inducing predicate (not (on ?container1 ?lid1)), which is a precondition for placing the object into the container. Only if the precondition is not met, the planner will schedule the action shown in Figure 16b, which has the required effect to meet the precondition.

```
(:action place-inside-211005224421
  :parameters (...)
  :precondition (and
    (in-reach ?position_sample_4)
    (not (empty-hand))
    (grasped-with ?cube1 ?grasp0)
    (not (on ?container1 ?lid1)))
  :effect (and
    (inside ?container1 ?cube1)
    (empty-hand)
    (not (grasped-with ?cube1
      ?grasp0))))
```

(a) Place cube into container

```
(:action place-on-211005224421
  :parameters (...)
  :precondition (and
    (in-reach ?position_sample_7)
    (not (empty-hand))
    (grasped-with ?lid1 ?grasp0))
  :effect (and
    (not (on ?container1 ?lid1))
    (empty-hand)
    (not (grasped-with ?lid1
      ?grasp0)))
)
```

(b) Remove lid from container

Figure 16: PDDL descriptions of actions that were automatically added to the symbolic description after exploration for task (c1) (see Table 1). Dependency-inducing predicate is highlighted in bold.

9. Initial Action Abstractions for HERON

This section introduces the action abstractions needed to formulate planning problems towards achieving the RI activities outlined in Section 2. The abstractions are amenable to a planning domain definition language (PDDL) formulation of the problem. Apart from the parameters each action takes as inputs, we list each action's preconditions and effects (see Section 3). Further, we describe how the action is intended to be grounded. Grounding refers to the method connecting an action to the real world. The actions are built upon the state abstractions presented in Section 6.

9.1. Traffic Cone Placement and Removal

- NAVIGATE ?target

  Parameters ?target a target pose for the cone

  Effects IN REACH target

  Description Navigate to a target location.

  Grounding Calls navigation framework such as the ROS navigation stack.

- PLACE ?cone ?target

  Parameters ?cone: a traffic cone, ?target a target pose for the cone

  Preconditions IN HAND cone, IN REACH target, CFREE cone target

  Effects not IN HAND cone, AT cone target, EMPTY HAND **Description** Place a cone at a target location.

  Grounding Calls arm motion planning framework such as the MoveIt to plan an arm trajectory to execute.

- PICK ?cone

  Parameters ?cone: a traffic cone

  Preconditions IN REACH cone, EMPTY _HAND

  Effects IN HAND cone, not EMPTY HAND

  Description Pick up a cone that is in reach of the robot.

  Grounding Calls arm motion planning framework such as the MoveIt to plan an arm trajectory to execute.

9.2.Filling Road Cracks

- NAVIGATE ?crack

  Parameters ?crack: a road crack

  Preconditions not AT crack

  Effects AT crack

  Description Navigate so that the crack can be operated on.

  Grounding Calls navigation framework such as the ROS navigation stack.

- CLEAN ?crack

  Parameters ?crack: a road crack

  Preconditions AT crack, not CLEAN crack

  Effects CLEAN crack

  Description Cleans up the crack, so that deposited material lasts longer.

  Grounding Specialized robot skill.

- DEPOSIT ?crack

  Parameters ?crack: a road crack

  Preconditions AT crack, CLEAN crack

  Effects DEPOSITED crack, not FLATTENED crack

  Description Deposits filling material on the crack.

  Grounding Specialized robot skill.

- FLATTEN ?crack

  Parameters ?crack: a road crack

  Preconditions AT crack, DEPOSITED crack

  Effects FLATTENED crack

  Description Compresses filling material on the crack to create a flat surface.

  Grounding Specialized robot skill.

9.3.Renewing Road Markings

- NAVIGATE ?marking

  Parameters ?marking: a segment of road marking

  Effects AT START marking

  Description Navigate to the start of a road marking and align the robot with it.

  Grounding Calls navigation framework such as the ROS navigation stack.

- CLEAN ?marking

  Parameters ?marking: a segment of road marking

  Preconditions AT START marking, FADED crack, not CLEAN marking

  Effects CLEAN crack, not AT START marking

  Description Cleans up the road marking, so that the color lasts longer.

  Grounding Specialized robot skill.

- PAINT ?marking

  Parameters ?marking: a segment of road marking

  Preconditions AT START marking, FADED marking, CLEAN marking

  Effects not FADED marking, not AT START marking

  Description Re-paints the road marking.

  Grounding Specialized robot skill.

9.4.Patching Potholes

- NAVIGATE ?pothole

  Parameters ?pothole: a segment of a pothole

  Preconditions not AT pothole

  Effects AT pothole

  Description Navigate so that the pothole segment can be operated on.

  Grounding Calls navigation framework such as the ROS navigation stack.

- CLEAN ?pothole

  Parameters ?pothole: a segment of a pothole

  Preconditions AT pothole, not CLEAN pothole

  Effects CLEAN pothole

  Description Cleans up the pothole segment, so that deposited material lasts longer.

  Grounding Specialized robot skill.

- DEPOSIT ?pothole

Parameters ?pothole: a pothole segment

Preconditions AT pothole, CLEAN pothole

Effects DEPOSITED pothole, not FLATTENED pothole

Description Deposits filling material on the pothole segment.

Grounding Specialized robot skill.

- FLATTEN ?pothole

Parameters ?pothole: a pothole segment

Preconditions AT pothole, DEPOSITED pothole

Effects FLATTENED pothole

Description Compresses filling material on the pothole segment to create a flat surface.

Grounding Specialized robot skill.

9.5. Inserting and Removing Road Tiles (RUPs/CUDs)

- NAVIGATE ?target

Parameters ?target a target pose for a RUP

Effects IN REACH target

Description Navigate to a target location.

Grounding Calls navigation framework such as the ROS navigation stack.

- PLACE ?tile ?target

Parameters ?tile: an RUP, ?target a target pose for the RUP

Preconditions IN HAND tile, IN REACH target, CFREE tile target

Effects not IN HAND tile, AT tile target, EMPTY HAND

Description Place an RUP at a target location.

Grounding Calls arm motion planning framework such as the MoveIt to plan an arm trajectory to execute.

- PICK ?tile

Parameters ?tile: an RUP

Preconditions IN REACH tile, EMPTY_HAND

Effects IN HAND tile, not EMPTY HAND

Description Pick up an RUP that is in reach of the robot.

Grounding Calls arm motion planning framework such as the MoveIt to plan an arm trajectory to execute.

10. Conclusions

This report constitutes deliverable D4.3 on representations for high level planning, presenting the work done for work package 4 titled “Motion and High-Level Planner for the automated HERON system”. It details the outcome of tasks 4.4 (“State abstractions for high-level

planning") and 4.5 ("Symbolic action representations for high-level planning"), the symbolic state and action models needed to implement task level planning for automating the road maintenance tasks in HERON.

In this vein, we gave an introduction into the road intervention tasks HERON targets, as well as into task-level planning methods and tools that will help achieving them.

Having established that state abstractions are highly task and environment dependent, we presented a framework for learning state abstractions from labelled data. This simplifies adapting and extending the set of state abstractions available to a symbolic planner, which is beneficial in dynamic and open-ended environments. Our work on state abstractions is complemented by initial predicates to be used for each task that will be encountered by HERON.

Building on the state abstractions, we turned to action abstractions. Similarly, open-ended environments may require extending the set of action models available for planning. To this end, we presented an algorithm for extending a symbolic action set automatically based on data collected during interactions of the agent in the environment. To initialize a symbolic planner for HERON, a set of basic actions for each road intervention task was introduced.

In future work, we will integrate the state and action abstractions into a planning system, which is ultimately to be integrated and tested on the real robot.

11. References

[1] Francois De Larrard, Thierry Sedran, and Jean-Maurice Balay. "Removable urban pavements: an innovative, sustainable technology". In: *International Journal of Pavement Engineering* 14.1 (2013), pp. 1–11.

[2] Erez Karpas and Daniele Magazzeni. "Automated Planning for Robotics". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2019), pp. 1–23.

[3] Charles R Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *arXiv preprint arXiv:1612.00593* (2016).

[4] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014, p. 9. url: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[5] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., Dec. 2017. url: https://proceedings. neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf (visited on 11/10/2021).

[6] Julian Förster et al. "Automatic extension of a symbolic mobile manipulation skill set". In: *Robotics and Autonomous Systems* 165 (2023), p. 104428.

[7] Levente Kocsis and Csaba Szepesvári. "Bandit based monte-carlo planning". In: *European conference on machine learning*. Springer. 2006, pp. 282–293.