Journal of Intelligent Manufacturing
https://doi.org/10.1007/s10845-024-02478-0

Adaptive acquisition planning for visual inspection in remanufacturing using reinforcement learning

Jan-Philipp Kaiser[1] · Jonas Gäbele[1] · Dominik Koch[1] · Jonas Schmid[1] · Florian Stamer[1] · Gisela Lanza[1]

Received: 16 March 2023 / Accepted: 3 July 2024

Abstract
In remanufacturing, humans perform visual inspection tasks manually. In doing so, human inspectors implicitly solve variants of visual acquisition planning problems. Nowadays, solutions to these problems are computed based on the object geometry of the object to be inspected. In remanufacturing, however, there are often many product variants, and the existence of geometric object models cannot be assumed. This makes it difficult to plan and solve visual acquisition planning problems for the automated execution of visual inspection tasks. Reinforcement learning offers the possibility of learning and reproducing human inspection behavior and solving the visual inspection problem, even for problems in which no object geometry is available. To investigate reinforcement learning as a solution, a simple simulation environment is developed, allowing the execution of reproducible and controllable experiments. Different reinforcement learning agent modeling alternatives are developed and compared for solving the derived visual planning problems. The results of this work show that reinforcement learning agents can solve the derived visual planning problems in use cases without available object geometry by using domain-specific prior knowledge. Our proposed framework is available open source under the following link: https://github.com/Jarrypho/View-Planning-Simulation.

Keywords Reinforcement learning · View planning · Acquisition planning · Inspection · Remanufacturing

Introduction

Growing global competition, increasing individualization, and volatile markets require production systems that adapt quickly and dynamically to changing influencing factors and requirements (Koren, 2010). To integrate new process technologies and functions to introduce new product variants and adjust processes to product variety, changeable production systems are needed (Mehrabi et al., 2000). As a vision, when considering operational processes, adaption is enabled at system runtime, without any human intervention, in the production process itself, and without any upstream planning process. This vision applies to all processes involved in the value creation of the products, such as, among others, assembly processes (Hu et al., 2008, 2011) and quality assurance (Schötz et al., 2017). Thus, agile production systems, utilizing autonomous production resources with high degrees of flexibility and adaptivity, are needed (Scholz-Reiter & Freitag, 2007), that can react to changes and uncertainty in the production environment (Arai et al., 2000). A field where autonomous systems are researched extensively is autonomous driving. Vehicles adapt their behavior independently to uncertain or new situations in autonomous driving. Machine learning methods provide the basis for solving various problems, such as detecting other traffic participants (Shen et al., 2023). An exemplary use case where autonomous resources need to react to uncertainty in

✉ Jan-Philipp Kaiser
Jan-Philipp.Kaiser@kit.edu

Jonas Gäbele
Jonas.Gaebele@student.kit.edu

Dominik Koch
Dominik.Koch@kit.edu

Jonas Schmid
Jonas.Schmid@student.kit.edu

Florian Stamer
Florian.Stamer@kit.edu

Gisela Lanza
Gisela.Lanza@kit.edu

[1] wbk Institute for Production Science, Karlsruhe Institute of Technology, Kaiserstraße 12, 76131 Karlsruhe, Germany

Published online: 27 August 2024

the industrial field is provided by the industrial process of Remanufacturing.

Remanufacturing is becoming increasingly important in the context of growing environmental awareness and dependence on central raw materials (Sundian, 2004). In remanufacturing, products originating from the use phase are disassembled. Individual components are reprocessed or replaced and reassembled (Daniel & Guide, 1997) to yield products with the same functionality as new products (Tolio et al., 2017). Due to the uncertainty about timing, quality, quantity, and variability of the returning products, processes involved in remanufacturing are mainly carried out manually (Kurilova-Palisaitiene et al., 2018). Thus, agile systems are necessary since conventional, rigidly automated systems do not provide the necessary flexibility and adaptivity to address the complexity resulting from the uncertainty in remanufacturing (Junior & Filho, 2012; Vongbunyong et al., 2015).

This challenge becomes evident during the initial inspection of the returned product, which precedes the actual process steps of remanufacturing. The initial inspection is carried out visually by a worker to identify defects on the products to sort out the ones that cannot be remanufactured (Errington & Childe, 2013). Schlüter et al. (2021) state that automating this process is challenging. Unlike existing approaches to visual inspection of linear production,

- occurring defects are highly diverse,
- they often occur anywhere on the product and
- the specific variant of the product is usually not known at the time of inspection.

An automated visual inspection system in remanufacturing for initial inspection must simultaneously be flexible and adaptive. Flexibility is required since numerous product variants must be thoroughly inspected. Inspection systems employing robot-guided acquisition systems can realize the necessary flexibility. Furthermore, adaptivity is needed as the respective product variant and associated geometric information such as *computer-aided design* (CAD) models are often not known at inspection time (Khan et al., 2021) and different product variants, especially those from different manufacturers, have differences regarding geometry and design. Usually, several acquisitions from different perspectives are necessary for a nearly complete acquisition and evaluation of an object. Together with the flexibility of robot-guided acquisition systems, *reinforcement learning* (RL) offers the possibility of learning adaptive inspection strategies for various variants of a given product, resolving the issue of solving the inspection task for various product variants despite non-available geometric models. By first training an RL agent in a simulation to perform an inspection task for a diverse set of existing products and their geometric models, the agent can capture the underlying characteristics required to complete the given inspection task. For new product types with unknown geometric models, the agent should then be able to generalize and perform the inspection task even for these product types.

Therefore, the visual inspection planning problem is first formulated as a sequential acquisition planning problem in the remainder of this work. This is followed by the detailed introduction of a simulation environment that can be used to train various differently modeled RL agents to solve visual acquisition planning problems.

The paper is structured as follows. In Section "Fundamentals and literature review", the requirements for inspection planning in remanufacturing are detailed. It is deduced why RL can be used to solve the inspection planning problem formulated as an acquisition planning problem. The methodical approach for adaptive initial visual inspection is presented in Section "Methodical approach for visual acquisition planning based on reinforcement learning". Results are presented and discussed in Section "Use case description and computational results.

Fundamentals and literature review

Initial visual inspection in remanufacturing

According to DIN 31051/ DIN EN 13306 (DIN 31051:2019-06 2019, DIN EN 13306:2018-02 2018), *inspection* is defined as the testing of the conformity of the relevant characteristics of an object by measurement, observation, or functional testing. The initial visual inspection can thus be categorized as the testing of the conformity of relevant characteristics by observation of a human being. Evaluating the reusability and value of an existing used product enables the identification of unusable used products. Errington and Childe (2013) consider visual inspection a central remanufacturing process. Ridley & Ijomah (2015) state that evaluating the used product through an initial visual inspection can save time for subsequent processes of up to 20%. Remanufacturers often specify core acceptance criteria (e.g., CoremanNet (2022)) for used products, which can be used to check in advance whether the return will lead to a refund of the deposit or purchase of the used product. This enables the supplier to sort out non-reusable products in advance. Based on said criteria, the remanufacturer can thus save time and money. The core acceptance criteria vary depending on the remanufacturer and product type. They provide information on permissible mechanical and electrical defects, acceptable corrosion, and the permitted degree of disassembly or missing components. In contrast to linear production, it is impossible to accurately predict where product defects will occur (Kurilova-Palisaitiene et al., 2018). Simultaneously, the possible occurring defects are manifold (Robotis et al., 2012) and often challenging to quantify. This applies, for

example, to the degree of corrosion. In particular, the diversity of defects poses a significant challenge. On the one hand, occurring defects can be of global nature and often appear on the entire product, e.g., corrosion or severe contamination. On the other hand, some defects result in geometric or color-based deviation from the defect-free product state and may be specific to individual product components, e.g., broken mounts or burnt electrical connectors. This manifold complexity of occurring defects results in a great challenge for implementing an efficient remanufacturing process and an automated inspection system. Therefore, the following requirements apply to automated systems for initial visual inspection.

- An inspection system is required to detect defects on the entire product surface.
- An inspection system that uses color-based information and geometric information to detect defects (e.g., images for corrosion detection and *three-dimensional acquisition systems* (3D systems) to detect dents) is required.

Acquisition systems that are able to acquire color images (*RGB* information) as well as depth (*D*) images are called RGBD systems. The acquired depth information of such RGBD systems can be used to calculate point clouds (XYZ information) represented in the acquisition systems' coordinate system. Robot-guided RGBD systems can meet the mentioned requirements and provide the necessary flexibility. Guiding such RGBD systems by a robot makes detecting and evaluating defects on the entire product surface possible. Thereby, the RGB information is used to identify the defects, and the sequential positioning of the robot is guided by the depth (D) or point cloud (XYZ) information. The problem of identifying a sequence of poses on the entire surface of an arbitrary object using dimensional (e.g., point clouds) object information can be formalized as an acquisition planning problem.

Definition of acquisition planning problems for visual inspection systems

Existing literature roughly divides visual acquisition planning problems into two different categories. A distinction is made between planning problems for which a priori information regarding the inspection target is unavailable and those for which this is the case. The former is widely referred to as the *next best view* problem (NBV problem) (Banta et al., 1995) in the literature. In contrast, the latter is known as the *view planning problem* (VPP) (Scott et al., 2003). Depending on the application, the output of the respective planning problem involves sensor poses (e.g., for structured light projection systems) or trajectories (e.g., for laser scanners) of a sensor (Peuzin-Jubert et al., 2021). Solutions to the VPP calculate a sequence of sensor poses or trajectories and can be solved before the system is deployed based on available a priori information (e.g., geometry models of a product to be inspected). Solutions to the NBV are search-based and compute the next best sensor pose or trajectory at runtime. In addition to the VPP and NBV, the *coverage planning problem (CPP)* additionally takes into account the coupled optimization of acquisition system poses and travel paths of the robot. For this work, the problems of NBV, as well as VPP and CPP, are defined as follows:

- **Next best view planning problem (NBV):**
  Banta et al. (1995) define the NBV as the next camera pose that will extract the most unknown scene information. One of the earliest mentions of the term *"next best view"* is from Connolly (1985), who focused on three-dimensional scene modeling. Further application areas can also be found, for example, in active object classification, where the assignment of an object to a particular class can only be made by taking several views into account Korbach et al. (2021).
  Within this work, the NBV problem is defined as finding the NBV of an acquisition system concerning an information criterion without considering global relations (e.g., minimizing the number of poses or minimizing travel distances of the robot).
- **View planning problem (VPP):** Scott et al. (2003) define the VPP as determining a suitable set of poses and associated imaging parameters for a specified object reconstruction or inspection task with a range camera and positioning system. Later works revised this definition to find the shortest possible view plan for a given environment and target object to achieve the planning goal in the shortest possible computational time (Scott, 2009). Many works also consider the path planning problem for finding the shortest paths of the robot end-effector between poses after solving the VPP. In most recent works, this is done after the VPP (Peuzin-Jubert et al., 2021).
- **Coverage planning problem (CPP):**
  Jing et al. (2018) therefore describe the coupled problem of trajectory planning and VPP and refer to it as CPP. This extension is necessary since, besides fulfilling the given planning goal in industrial use cases, the fastest possible execution is also of interest. The optimal solution of the CPP may thus be different from that of the VPP (Scott et al., 2003) since the execution time of the optimal view plan might be significantly higher compared to a slightly different view plan with comparable but slightly worse performance regarding the planning goal.

When analyzing the different variants of visual acquisition planning problems, it becomes evident that they can be formulated as a sequential decision process. Given an envi-

ronment and a target object, the task is to define successive sensor poses to meet a given inspection target (complete object coverage for VPP with possibly simultaneous consideration of the robot's travel costs when considering CPP).

Foundations of reinforcement learning

RL is suitable for solving sequential decision problems (Sutton & Barto, 2018). One or more agents interact with the so-called environment via actions. During an interaction, an agent perceives the current state of the environment and performs an action based on this information. The agent receives a reward based on the contribution of its action to achieving an overall goal. The agent's objective is to maximize this reward in the long term and thus fulfill the goal. Thus, RL differs fundamentally from supervised and unsupervised machine-learning approaches (Panzer & Bender , 2021), as agents are not used to predict individual data but have to learn a strategy for solving sequential problems. This is often achieved by trial-and-error-based, continuous interaction with the environment and a predefined reward signal. Successful RL applications are found in games (Mnih et al., 2013; Silver et al., 2017; Berner et al., 2019), where superhuman performance could be achieved. Other application areas include robotics (Kober et al., 2013) or production system control (Panzer & Bender , 2021; Kuhnle et al., 2019, 2021).

In RL, the problem to be solved must be a so-called *markov decision process* (MDP), which represents the formalization of sequential decision-making processes (Sutton & Barto, 2018). A MDP is described by a set of possible *states* $\mathcal{S}$ of the environment, a set of possible feasible *actions* of the agent $\mathcal{A}$, a *reward function* $\mathcal{R}$, and a probabilistic description of the *environment dynamics* $\mathcal{P}$ (van Otterlo et al., 2012). At each step in time, an agent performs an action based on its problem-solving *strategy* π. This action transforms the environment's state into a subsequent state according to the environmental dynamics of the underlying MDP. The agent receives a numerical reward based on the action it performed. The amount of the *reward* $r_{t+1} \in \mathcal{R}$ is calculated using the reward function and depends on the *action* $a_t \in \mathcal{A}$ taken at the time t, which transferred the environment from *state* $s_t \in \mathcal{S}$ to state $s_{t+1} \in \mathcal{S}$. The reward function has to be chosen so that actions that are useful for solving the decision problem are selected at each point in time. Mathematically formulated, the agent must learn an *optimal strategy* π^* that maximizes the sum of the discounted rewards (see Eq. 1), also called the long-term *return* G_t (Sutton & Barto, 2018).

$$G_t = r_{t+1} + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

The *discount factor* $\gamma \in [0, 1]$ represents a parameter that indicates the farsightedness of the strategy learned by the agent. By choosing $\gamma = 0$, the agent only learns a policy π that determines actions that lead to an immediate reward. When using $\gamma = 1$, future rewards are not discounted, and the agent learns a farsighted strategy (Sutton & Barto, 2018). However, the higher γ is, the more difficult it is to correctly estimate rewards far in the future based on the current state, leading to a more demanding learning problem for the agent.

RL agents are trained in simulations first to obtain cost-efficient and fast training results (Kober et al., 2013). They are then fine-tuned or directly deployed, where a significant body of research addresses the gap between simulation and reality when trying to integrate trained agents in actual use cases (Peng et al., 2018). Classical RL algorithms have solved many problems but often require significant feature engineering and may struggle with high-dimensional or continuous state and action spaces. In contrast, deep RL approaches allow agents to learn directly from raw sensory input and can automatically discover valuable representations of the environment (Mnih et al., 2015). This makes them well-suited for problems that involve complex, high-dimensional state and action spaces, such as playing complex games, controlling robots, or navigating complex environments (Lillicrap et al., 2015). Therefore, deep RL has become a popular and promising research area with the potential to advance artificial intelligence in many domains. Visual acquisition planning problems represent such complex problems, as the input space may consist of images or point clouds, and the output space may correspond to poses of an acquisition system in continuous six-dimensional space when no further simplifications of the problem are made.

Approaches for the solution of visual acquisition planning problems using machine learning

In contrast to classical analytical methods, machine-learning techniques offer the possibility of approaching visual acquisition planning problems from a data-driven perspective. This offers two key advantages. First, it may be possible to implement generalizing approaches for the problem of acquisition planning that can adapt to changing objects. These may learn to deal with objects whose surface model is unavailable and thus mimic human operators setting up acquisition routines. Furthermore, constraints to be considered (e.g., a spatially varying pose of the object to be acquired or constraints in the motion space of the robot resulting in restrictions for the solution of the planning problem itself) create additional complexity that can potentially be handled more efficiently with data-driven and learning methods.

In two consecutive works, Mendoza et al. (2020) and Vasquez-Gomez et al. (2021) address the NBV problem using supervised learning. Based on an occupancy grid, Mendoza

et al. (2020) determine the NBV based on predefined poses. Vasquez-Gomez et al. (2021) resolve the limitation to fixed poses and consider the problem as a regression problem for positioning in three-dimensional space. The authors showed that both approaches provide a significantly faster computation of the NBV compared to classical approaches but perform slightly worse in the final object coverage. Zeng et al. (2020) takes a comparable approach. In contrast to occupancy grids, the authors use point clouds as network input to determine the NBV. For this purpose, the authors develop the so-called *point cloud next best view network* (PC-NBV-Net) based on the PointNet architecture. As in the work of Mendoza et al. (2020), the NBV is classified from a discrete set of poses using the PC-NBV-Net. The authors showed that their approach yields promising results even for previously unknown objects and thus exhibits a certain generalizability. Other relevant work uses neural networks for scene completion and subsequent planning (Monica & Aleotti, 2021), for determining one (Ashutosh et al., 2022) or multiple (Pan et al., 2022) poses for optimal object reconstruction.

RL has also been successfully applied to acquisition planning and related problems. Deinzer et al. (2009) and Korbach et al. (2021) deal with active object recognition. In these works, strategies for choosing the NBV that provides the highest information content concerning classifying the object at hand into a set of classes were learned. A pioneering work in the area of solving the VPP using RL is that of Devrim Kaba et al. (2017). For the first time, the authors formulate the VPP as an MDP and solve it using RL. Further work has taken up these ideas. Similar to works in the supervised setting, Potapova et al. (2020) propose an occupancy grid of the object as the state of the environment and solve the VPP using a sequential selection of defined poses by the agent. In contrast, Landgraf et al. (2021) and Devrim Kaba et al. (2017) model the state of the environment in terms of the current sensor pose. The agent must select the NBV to solve the VPP in both cases. In the latter cases, however, the agent does not receive information about the object's geometry to be acquired.

Research deficit and contribution

Existing methods for solving the problem of acquisition planning using supervised learning mainly address the problem of NBV planning (c.f. Mendoza et al. (2020); Vasquez-Gomez et al. (2021); Monica and Aleotti (2021); Pan et al. (2022); Zeng et al. (2020)). Due to the requirement of direct feedback in the form of an optimal pose as the label, these solution methods are limited to finding the best possible next acquisition step. The evaluation of the quality of sequential decisions is usually missing. Thus, addressing the solution of the VPP is only given to a limited extent, and the integration of the travel distances of a robot-guided acquisition system and, thus, the solution of the CPP is not given.

Approaches using RL offer the possibility to fulfill this requirement. However, a closer analysis of existing work reveals that different approaches to problem modeling exist. In particular, the following research deficits can be deduced:

1. The modeling of state and action space (input and output space) and the definition of the reward signal, which defines the agent's objective, varies greatly between existing works and is not part of systematic investigations in existing works.
2. Substantial simplifications of the problem have been made in the past. For example, the use of low-resolution occupancy grids (c.f. Potapova et al. (2020)) or sensor poses (c.f. Landgraf et al. (2021)) as state space represents a substantial simplification of object geometries. Such simplifications can lead to only locally optimal strategies or completely prevent the problem's solution, especially for objects with complex geometries.
3. RL approaches for the solution of the VPP that are free of an object model at runtime, i.e., they do not require knowledge of the geometry of the product to be inspected, have yet to be considered. The next pose of the acquisition system has to be determined at system runtime without knowledge of the product at hand and without preplanning using an available object geometry model. Landgraf et al. (2021), Potapova et al. (2020), and Devrim Kaba et al. (2017) all assume the object model to be known since the modeling of their state and/or action space depends on said model. These approaches thus make use of a preplanning stage prior to deployment of a then calculated and fixed view plan. In Remanufacturing, however, this information can not be assumed to be known. Products must be inspected without any given model, which makes a model-free approach necessary.

To address the research gaps deduced, the present work expands the state of research through its holistic approach. Our goal is to:

- Formulate and solve the acquisition planning problems as generally as possible.
- Implement and compare various different modeling alternatives of RL agents for the solution of the acquisition planning problems introduced in Section "Definition of acquisition planning problems for visual inspection systems". Therefore, we explicitly focus on a model-free approach.

To this end we:

1. Develop an RL simulation framework consisting of a simulation to model the acquisition process and an instance

(in our case, RL agent) to determine the next pose to capture an object to be inspected. A special focus is laid on the simplicity and computational efficiency of the simulation framework so that a quick simulation and, thus, a quick training and comparison of differently trained instances is possible. This is needed since RL, to this day, is still sample inefficient and sensitive to varying hyperparameters.
2. Maintain a strict modular separation between the actual acquisition process and the determination of the next pose of the acquisition system by the instance. This allows for fast exchange and evaluation of differently configured RL agents or other instances to determine the next pose.
3. Employ a simulation to model the acquisition process, replicating the essential characteristics of an acquisition process using a three-dimensional acquisition system. To this end, essential parameters of the simulated acquisition system, such as its optimal working distance, are parameterized to allow for a fast switch between different configurations or variants of real acquisition systems.
4. Systematically deduce and model various alternatives for the state, action, and reward design of RL agents for acquisition planning problems as generally as possible in a model-free setting. We show and discuss common design decisions and problems and aim to keep the design choices as general as possible so that these design decisions can be reused in the future and other works to solve acquisition planning problems.
5. Aim for an agent implementation in a modular way. Individual function modules can be used to configure and train agents of different modeling alternatives. Thus, these modeling alternatives can be evaluated concerning their applicability to the introduced planning problems, and an optimal agent configuration can be found.

Methodical approach for visual acquisition planning based on reinforcement learning

This section provides an overview of the proposed RL simulation framework (Section Overview of the reinforcement learning simulation framework"), followed by a detailed introduction to the modeling of the scan simulation environment (Section "Modelling of the scan simulation environment) as well as the agent (Sects. "Modelling of the agent interface and "Agent modeling").

Overview of the reinforcement learning simulation framework

The RL simulation framework developed in this work is depicted in Fig. 1. It is composed of a scan simulation environment (see Section "Modelling of the scan simulation environment") as well as an agent interface (see Section "Modelling of the agent interface") and the agent itself (see Section "Agent modeling"). This structure allows the easy interchangeability of individual modules to analyze different modeling alternatives and the learned strategies of the defined RL agents to solve the visual acquisition planning problems. The simulated environment encapsulates all functionalities needed for three-dimensional object acquisition by using a sensor and object model in conjunction with a scanning module to simulate the acquisition process. The functionalities necessary for RL are defined in the agent interface's action, state, and reward modules to enable interaction between the agent and the scan simulation environment. The agent module consists of the definition of the agent itself. Due to standardized interfaces, replacing the simulation module with the real acquisition system at a later stage is straightforward.

Modelling of the scan simulation environment

The simulation environment contains a scanning module that simulates the acquisition of an object by a three-dimensional acquisition system. An object model describes the object, and a sensor model describes the optical three-dimensional acquisition system. The scanning module also aggregates the acquisition results, which are point clouds, of the sequentially performed acquisition steps. The individual functionalities are described in detail below.

Sensor model and object model

The sensor model reproduces the functionality and properties of the three-dimensional acquisition system used. The relevant configuration options of the simulated acquisition system result from modeling the field of view as a frustum and the specification of the resolution. The parameters to formally describe the simulated acquisition system thus include the following:

- Resolution of the acquisition system
- Aperture angles of the frustum (field of view)
- Near and far bounds of the frustum (operating range of the acquisition system)

The object model used in the simulated environment is specified via the *stereolithography* (STL) format. Thereby, the object is represented only by its surface in triangular facets. The representation of the object with triangular facets enables computationally efficient ray tracing procedures, which are used by the scanning module and are explained below.

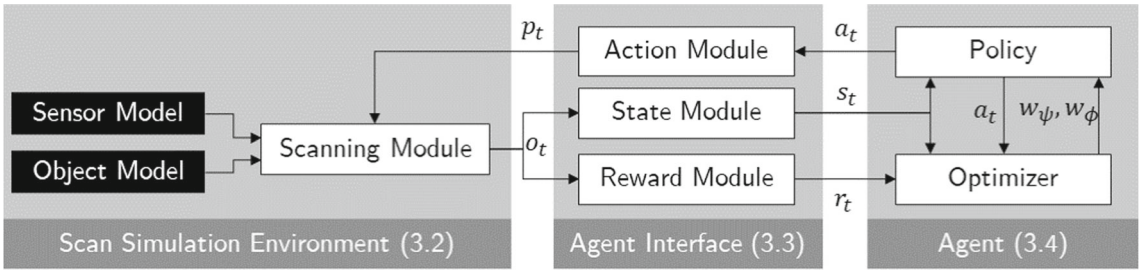

Fig. 1 Structure of the RL simulation framework

Scanning module

The scanning module connects the sensor model and the object model. The scanning module utilizes the Python package trimesh (Dawson-Haggerty et al. (2019)) to enable efficient ray tracing to emulate the three-dimensional acquisition of the object. Compared to existing frameworks (e.g., Blender), trimesh offers the advantage that only the actual process of acquiring a point cloud is simulated by ray tracing. Restrictions (e.g., near and far bounds) can be integrated into the framework very easily. Furthermore, no additional add-ons are required for implementation (as with Blender), and the overhead associated with available robot simulations (e.g., Gazebo) is avoided. In doing so, the scanning module receives a *acquisition system pose* $p_t = (x, y, z, \alpha, \beta, \gamma)$ from the RL agent. For x, y, and z, Cartesian coordinates indicate the *position*, while α, β, and γ are Euler angles, representing the *rotation* of the frustum. Afterward, a virtual acquisition system is placed in space based on the given pose p_t, and an acquisition is simulated. Depending on the resolution and aperture angles of the sensor model, rays are defined starting from the focal point. The coordinate of each first intersection point of the rays with the object defines a point in space in a global coordinate system. After simulating the acquisition of the object with the scanning module based on the acquisition system pose p_t, the *point cloud* P_t, as a sum of all intersection points of the rays with the object, is returned. Given a pose p_t where the object does not lie in the frustum of the acquisition system, P_t is empty. It has to be noted that there are initially no restrictions concerning the positioning of the simulated acquisition system in this work. This is contrary to reality since, for example, with robot-guided acquisition systems, the reachability of a certain pose of the acquisition system must always be verified.

The scanning module also bundles all information relevant to the agent through *observation data* o_t. o_t represents all information of the acquisition process up until the current interaction step t. At the beginning of each acquisition process, an object model (STL model) to be inspected is fed into the simulated environment and stored in the observation data. Additionally, at the beginning of an acquisition process, the *ground truth point cloud* P_{GT} is calculated and stored in the observation data as well. P_{GT} is generated by evenly sampling n_{GT} *ground truth points* on the object model using a voxel grid approach as explained in Section "State module". The scanning module then serves to collect all information gathered by the sequential acquisitions of the simulated environment up until interaction step t and stores them in the o_t. This observation is then used to derive the state s_{t+1} given to the agent to select the following action a_{t+1} at the interaction step $t + 1$. In each acquisition step t of the agent with the environment, the scanning module receives the next desired pose p_t, simulates an acquisition that results in a point cloud P_t, and triggers an update of the observation data. Next to the object model and the ground truth point cloud P_{GT} the following information given in the list below is computed and stored in the observation data o_t as well as updated after each acquisition:

- Number of acquisitions t in the current acquisition process.
- List of acquisition system poses $p_1, \ldots, p_t$ of the current acquisition process and the corresponding acquired point clouds $P_1, \ldots, P_t$.
- *Total point cloud* $P_{cov,t}$ acquired in an acquisition process up to interaction step t.
- Total point cloud $P_{cov,t-1}$ acquired up to interaction step $t - 1$.
- *Inverted point cloud* $P_{inv,t-1}$ at interaction step $t - 1$ which is calculated based on the ground truth point cloud P_{GT} subtracting the point cloud $P_{cov,t-1}$ acquired up until interaction step $t - 1$.

Based on this information, the corresponding state representation can be calculated in the state module, which is explained in Section "State module".

An example of two consecutive acquisitions is given in Fig. 2. In the first acquisition with pose, p_1 the point cloud $P_1 = P_{green}$ is acquired. In this case, $P_{cov,1} = P_{green}$ holds. Subsequently, the point cloud $P_2 = P_{red}$ is acquired with the pose p_2. Thus, $P_{cov,2}$ equals the union of P_{green} and P_{red} resulting in $P_{cov,2} = P_{green} \cup P_{red}$. In addition, the ground

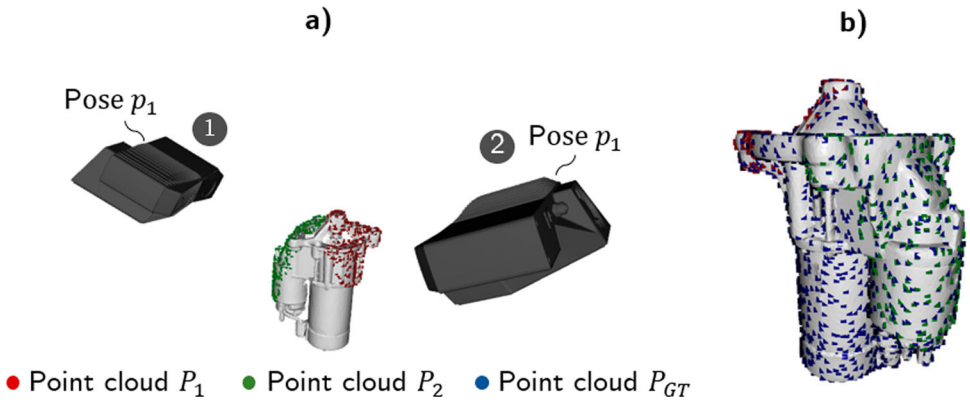

Fig. 2 Visualization of the acquisition process. **a** Two subsequent acquisitions that result in point clouds $P_1 = P_{green}$ and $P_2 = P_{red}$. **b** A close-up shows the ground truth point cloud P_{GT}, which contains different points than P_{green} or P_{red} on the whole surface of the object, in blue

truth point cloud P_{GT} is visualized in blue, which plays an essential role in the object coverage and reward calculation in Section "Reward module". The agent's task is now to choose the poses p_i of the acquisition system so that, for instance, the largest possible object area, which has not yet been covered with previous acquisitions, is covered. Figure 2 shows, for example, that the object coverage by P_{green} is significantly larger than P_{red}. p_1 is therefore a better view point than p_2 in this case. However, it should be noted that this is only the case if previous acquisitions did not already cover the object areas covered by P_{green}. This would reduce the proportion of the area newly covered by pose p_1. These relations have to be learned by the agent, and at each time step, the pose has to be determined, which is optimal for solving the visual planning problem at hand.

Definition of an episode

In RL, an episode is a state, action, and reward transition sequence. An episode ends in a terminal state s_T. In the context of acquisition planning, an episode is equivalent to the acquisition process of one object and is thus part of the environment. In this work, there are two options implemented that end an episode. Firstly, to complete the inspection process successfully, a threshold value of the surface coverage is defined, which must be achieved to complete an episode successfully. A basis for estimation of the surface coverage of the object in an episode is given in Section "Reward module". A state in which this threshold has been reached is called a *terminal state* s_T. Secondly, limiting the maximum number of allowed acquisitions is reasonable due to the industrial context. For this reason, a limit of steps is introduced, which terminates an episode when reached. For the definition considered in this work, reaching a surface coverage of the object of 90 % leads to a terminal state, and reaching a maximum permitted number of ten acquisitions forces the episode to end without reaching a terminal state.

Modelling of the agent interface

The agent interface transfers information provided by the scan simulation environment into a representation that can be interpreted by the agent (see Section "State module"). Furthermore, it translates selected agent actions into an acquisition step interpretable for the scan simulation environment (see "Action module" section). Besides, the agent interface also calculates the rewards for the learning agent (see Section "Reward module").

State module

In the state module, the information stored in the overall observation data o_t is used to deduce the state s_t given to the agent to determine the acquisition system's next pose p_{t+1}. To ensure the transferability of the simulated system to a real scenario where no object model is available, it is important that only data that is also available to the agent in the said scenario is integrated into the calculation of the state s_t. Modeling the state vector s_t is thus a fundamental task. This implies that, for example, after an initial acquisition, only the point cloud P_0 of this acquisition may be used to deduce state s_0 and passed to the agent. Further information that can be extracted from the object model (e.g., the shape) cannot be used to model the state.

Regarding acquisition planning problems, we consider two approaches to provide the agent with state information s_t at interaction step t. In the first option, we provide the agent with information based on the surface points of the object that acquisitions have already acquired until interaction step t. In the second option, we provide the agent with information based on which surface points have already been acquired until interaction step t and which surface points still need to be acquired in the rest of an episode to fully capture the entire object surface. To realize this, we employ point clouds to represent the state. In contrast to low-resolution occupancy grids, point clouds can represent object geometries in greater detail.

The first option for state coding presented in this work is based on the total point cloud $P_{cov,t}$ acquired up to the current interaction step t of the agent with the environment. Thereby, $x_{t,m}$, $y_{t,m}$ and $z_{t,m}$ denote the X, Y, and Z values of a point m of the point cloud $P_{cov,t}$. The state in the form of the *coverage matrix* $s_{t,cov}$ (see Eq. 2) captures the maximum available information about the present unknown object. Thus, the agent must select a pose for the subsequent acquisition that maximizes the number of newly acquired points on the object surface that are not included in $s_{t,cov}$ and, therefore, have yet to be acquired.

$$s_{t,cov} = \begin{bmatrix} x_{t,1} & y_{t,1} & z_{t,1} \\ \dots & \dots & \dots \\ x_{t,2048} & y_{t,2048} & z_{t,2048} \end{bmatrix} \tag{2}$$

Since $P_{cov,t}$ is generated based on point clouds of several consecutive acquisitions, a voxel downsampling is carried out before calculating $s_{t,cov}$. This has two key advantages. First, voxel downsampling provides a point cloud $P_{cov,t}$ with equal density distribution. Second, voxel downsampling enables reducing the size of the point cloud to a fixed value. This is necessary since the agents employ neural networks (see Section "Agent modeling") that require a fixed state size s_t as inputs. In the present work, $P_{cov,t}$ is therefore reduced to a size of 2048 points. The number of points was chosen to correspond to a size that is also frequently chosen as a reference in other works, such as in, for example, Achlioptas et al. (2018), Wang et al. (2020) and Wen et al. (2020). It has to be noted that there is no general rule in choosing the number of points in the point cloud. The number of points should be chosen as low as possible to reduce calculation time but as high as necessary to preserve the object's shape.

The second option for modeling the state s_t is a binary state encoding. It is based on the assumption of a geometric model of the object to be inspected. The agent receives a *binary state matrix* $s_{t,bin}$ (see Eq. 3) at the beginning of each acquisition step t. Each point in $s_{t,bin}$ corresponds to a point in the ground truth point cloud P_{GT}. Thus, in analogy to the first state encoding $s_{t,cov}$, the number of points n_{GT} in P_{GT} is set to 2048 in this work. The last column encodes whether a point has been seen in the current acquisition process.

$$s_{t,bin} = \begin{bmatrix} x_{t,1} & y_{t,1} & z_{t,1} & b_{t,1} \\ \dots & \dots & \dots & \dots \\ x_{t,2048} & y_{t,2048} & z_{t,2048} & b_{t,2048} \end{bmatrix} \tag{3}$$

When using $s_{t,bin}$, the following applies with $m \in \{1, \dots 2048\}$:

$$b_{t,m} = \begin{cases} 1 & (x_{t,m}, y_{t,m}, z_{t,m}) \in P_{cov,t} \\ 0 & (x_{t,m}, y_{t,m}, z_{t,m}) \notin P_{cov,t} \end{cases} \tag{4}$$

If a so far unknown point has been acquired during an acquisition step, $s_{t,bin}$ is updated and given to the agent to determine the pose for the subsequent acquisition. It should be noted that state encoding $s_{t,bin}$ explicitly requires an object model in the application phase. However, this is not present in the cases considered for model-free view planning. Nevertheless, point cloud completion approaches (e.g., Yuan et al. (2018) and Huang et al. (2020)) are already able to estimate a rough model of the complete object based on a partial present point cloud. These approaches can be used in combination with the present modeling, which is therefore considered and evaluated in the context of this work. This is also why the inverted point cloud $P_{inv,t-1}$ is stored in every timestep, since this point cloud may serve as a ground truth target for such approaches in future works.

Action module

The action module generally receives an n-dimensional *action vector* a_t as the agent's output in each interaction step with the environment. Each component $a_{t,i}$ of the action vector seen in Eq. 5 has values between -1 and 1. This range is defined by the activation function, in this case, tangens hyperbolicus, of the last layer of the neural network used to approximate the agent's policy. See Section "Agent modeling" for more details on the network used.

$$\begin{aligned} a_t &= (a_{t,1}, \dots, a_{t,n})^T \\ & a_{t,i} \in [-1, 1] \quad \text{with} \quad i \in \{1, \dots n\} \end{aligned} \tag{5}$$

The action module then maps a_t to an acquisition system pose p_t to be processed by the simulation environment. This is necessary since the pose p_t is defined concerning its position in Cartesian coordinates, and the orientation has to be specified using Euler angles. However, when specifying the action vector a_t as the agent's output, other coordinate definitions or the incorporation of prior knowledge can be helpful, as explained in the remainder of this section.

General modeling alternatives of the action vector

This work uses Cartesian or spherical coordinates to model the position component of the action vector a_t. In the case of spherical coordinates, the first three entries $a_{t,1-3}$ correspond to the *azimuth angle* φ, *polar angle* θ and the *radius* r. A potential advantage of defining the components of the action vector in spherical coordinates lies in the dependence of the distance of the acquisition system from the object to be detected on only the parameter r. This suggests that the RL agent can easily learn the positioning of the acquisition system from the object at an optimal working distance. In the second case, the Cartesian coordinates, the first three entries $a_{t,1-3}$ of a_t correspond to the x, y, and z coordinates of the next pose of the acquisition system. In both cases, the origin of the coordinate system is located in the center of gravity

of the object to be acquired. There are also different options for modeling the components of the action vector a_t, which define the orientation of the acquisition system. This work considers the acquisition systems' orientation via the representation of the entries in terms of Euler angles. In addition to the three entries for the position, the action vector has two more entries representing the Euler angles α and β. In the present work, the angle γ (rotation around the longitudinal axis) is assumed to be constant and is not supposed to be learned by the agent. This is based on the assumption that the influence of γ on the acquired point cloud is negligible due to the nearly rotationally symmetric frustum. Therefore, $\gamma = 0$ is used to calculate the pose of the acquisition system. Another possibility is the representation of the orientation with Quaternions. However, we are not considering using Quaternions for now due to simplicity.

In order to position the acquisition system in the global coordinate system, when specifying the action vector a_t with spherical coordinates, the action module converts them into Cartesian coordinates. In addition to the functionality of mapping between different coordinate systems, the action module also handles the adjustment of the value ranges. Due to dealing with actions output from the agent by neural networks (see Section "Agent modeling"), the value ranges of all the components $a_{t,i}$ are defined to be limited to the range between -1 and 1. These must then be mapped to valid ranges of the desired representations. For spherical coordinates, this implies that a mapping of the value range $[-1, 1]$ into the value range $[0, 2\pi]$ for the azimuth angle φ as well as into the range $[0, \pi]$ for the polar angle θ has to be performed. An overview of all implemented variants for modeling the RL agents' action vector can be seen in Table 1.

Integration of prior knowledge

By integrating prior knowledge, individual degrees of freedom of the agent can be restricted. An overview of the restriction of the degrees of freedom of the agents output used in our framework can be found in Table 2. Ideally, agents can be trained faster since the solution space is restricted, so the discovery or exploration of non-optimal strategies is excluded from the beginning. In the case of spherical coordinates, limiting the radius r to one fixed value, representing the optimal working distance of the acquisition system, is possible. This leads to the action modeling variant A_{2S2R} using M^{r}_{wpk}. Concerning orientation, prior knowledge can also be applied to ensure that the acquisition system is always oriented toward the object's center of gravity using $M^{\alpha,\beta}_{wpk}$. Thus, it is possible to have the RL agent learn the position of the acquisition system, with the orientation always being specified by the integrated prior knowledge. The number of entries of the action vector a_t as the output of the agent can thus be reduced to the three parameters (A_{3S0R}) or two parameters (A_{2S0R}) when using a fixed radius r (M^{r}_{wpk}) using $M^{\alpha,\beta}_{wpk}$ to define the orientation. Besides, an intermediate solution with the action variant A_{3S2R_lim} exists. In this case, as with the previously mentioned action variants, the rotation is determined so that the object's center of gravity is focused. The agent can choose an *angular deviation* $\Delta\alpha, \Delta\beta$ based on these calculated angles to allow a small degree of rotational freedom. The prior knowledge used is $M^{\Delta\alpha,\Delta\beta}_{wpk}$ Analogous approaches also result in modeling the action variants A_{3C0R} and A_{3C2R} using Cartesian coordinates.

Table 1 Implemented variants for modeling the actions of the RL agent

Abbreviation	Position	Rotation	DoF
A_{2S0R}	$\varphi = a_1$ $\theta = a_2$	–	2
A_{3S0R}	$\varphi = a_1$ $\theta = a_2$ $r = a_3$	–	3
A_{2S2R}	$\varphi = a_1$ $\theta = a_2$	$\alpha = a_3$ $\beta = a_4$	4
A_{3S2R_lim}	$\varphi = a_1$ $\theta = a_2$ $r = a_3$	$\Delta\alpha = a_4$ $\Delta\beta = a_5$	5
A_{3S2R}	$\varphi = a_1$ $\theta = a_2$ $r = a_3$	$\alpha = a_4$ $\beta = a_5$	5
A_{3C0R}	$x = a_1$ $y = a_2$ $z = a_3$	–	3
A_{3C2R}	$x = a_1$ $y = a_2$ $z = a_3$	$\alpha = a_4$ $\beta = a_5$	5

The lowercase numbers in the abbreviation of the action definition denote the degrees of freedom (DoF) followed by lowercase letters, which denote either the coordinate system (S for spherical and C for Cartesian) or the rotation R the degrees of freedom are allocated to

In addition to integrating prior knowledge into the action modeling, prior knowledge can also be incorporated into mapping the agent's output values to the actual values of the pose to be approached by the acquisition system. This can be done by adjusting the value ranges into which the output values of the agent are mapped. An example is $M^{x,y,z}_{wpk}$, which restricts the x,y and z values output by the agent to be too near to the object center and too far away from it.

Reward module

In order to learn a desired behavior that solves the problem at hand as well as possible and to continuously evaluate the actions performed, an RL agent receives feedback in the form of a numerical reward r_t. Concerning the reward frequency, we deploy two different approaches in this work. First, we use a dense reward signal given to the agent after

Table 2 Overview of integration approaches for prior knowledge

Parameter	Mapping	Value range
r	M^{r}_{npk}	$[0-100]$
r	M^{r}_{wpk}	47
x, y, z	$M^{x,y,z}_{npk}$	$[0-100]$
x, y, z	$M^{x,y,z}_{wpk}$	$[20-50]$
α, β	$M^{\alpha,\beta}_{npk}$	$\alpha \rightarrow [0, 2\pi]$ $\beta \rightarrow [0, \pi]$
α, β	$M^{\Delta\alpha,\Delta\beta}_{wpk}$	$\alpha \widehat{=} \Delta\alpha \rightarrow [-10°, 10°]$ $\beta \widehat{=} \Delta\beta \rightarrow [-10°, 10°]$
α, β	$M^{\alpha,\beta}_{wpk}$	fixed on object center

M^{X}_{npk} denotes mapping alternatives without integration of prior knowledge, whereas M^{X}_{wpk} denotes alternatives with the additional integration of prior knowledge

each interaction step of the agent with the environment. Second, sparse rewards are used. In this case, a reward is given after an episode reaches the terminal state s_T at time step T, as explained in "Definition of an episode" section. Dense rewards offer the advantage of making it easier for the agent to learn a strategy since it receives a reward after each step. In this case, the challenge is to evaluate the selection of individual action in such a way that it contributes to fulfilling the long-term goal. An insufficient evaluation can make it impossible for the agent to find the optimal solution to the problem. In contrast, sparse rewards grant the achievement of goals at the end of an episode. The agent receives a reward only once in an episode but with a direct indication of whether the agent has fulfilled the goal.

In general, the reward r_t is one numerical value. However, for the planning problems mentioned in chapter "Definition of acquisition planning problems for visual inspection systems" section, there exist partly competing evaluation measures, which can be used to evaluate the fulfillment of the objective of the agent. When modeled as a MDP to be solved by a RL agent, these quantities must be combined in an adequate way. We define the relevant evaluation measures for the individual planning problems as follows:

- Acquired surface coverage of the object with one acquisition (NBV) or after an episode (VPP, CPP)
- Number of acquisitions required for complete or near complete object coverage (VPP, CPP)
- Length of traveling distance between successively defined poses (CPP)

The choice of evaluation measures is based on the most fundamental evaluation measures for visual planning problems. The primary goal of a VPP is the complete coverage of the object surface (evaluation measure *acquired surface coverage*) (Peuzin-Jubert et al., 2021). Secondary goals are, for example, a view plan that is as short as possible (evaluation measure *number of acquisitions*) and, in the industrial environment, the minimization of the travel time of the acquisition system between acquisitions (evaluation measure *minimization of travel distances*). Further evaluation variables depend on the area of application of the VPP solution. For metrological applications, the accuracy of the acquired point clouds or their resolution may be important (Scott, 2009). However, we aim to first address the fundamental evaluation measures mentioned and leave the evaluation of secondary ones to future research.

Approximation of surface coverage

The object coverage is an optimization variable to be maximized by as few acquisitions as possible. It is, therefore, also a fundamental quantity in calculating the reward itself. In order to calculate the covered surface based on a point cloud acquired by one acquisition, an estimation is necessary. First, the point cloud $P_{cov,t}$ is acquired until interaction step t is downsampled to the same point density (voxel grid size) as the ground truth point cloud P_{GT} using a voxel grid approach. The *relative surface coverage* COV_t can then be estimated by the n_t acquired points on the ground truth point cloud P_{GT} by the acquisition system until interaction step t and the maximum number of possible ground truth points n_{GT} to be acquired. This is done by assigning each point of the acquired point cloud $P_{cov,t}$ until interaction step t to a point of the ground truth point cloud P_{GT} if its nearest neighbor in P_{GT} falls below a certain threshold distance ϵ. The parameter ϵ is thereby dependent on the point cloud density of $P_{cov,t}$ and P_{GT} and consequently the point distances between them. Therefore, the parameter ϵ was chosen heuristically, resulting in $\epsilon = 0.2$. Note, a higher density of $P_{cov,t}$ and P_{GT} should result in choosing a lower ϵ. The relative object coverage COV_t can then be estimated using Eq. (6).

$$COV_t \approx \frac{n_t}{n_{GT}} \tag{6}$$

Next to the percentage COV_t of the object surface already covered up until interaction step t, the percentage of *remaining object surface* to be covered $COV_{rem} = 1 - COV_t$ is important for the reward calculation.

Approximation of the length of traveling distance

For industrial application cases, the length of the traveling paths is a relevant evaluation measure. It significantly influences the execution time of an acquisition plan and, therefore, has to be included when the reward calculation is done. In this work, we intentionally avoided integrating an environment model and a robot model to address the effects of state, action, and reward modeling. Thus, consideration of the reachability of individual poses is impossible, and a computation of the travel distances is only approximate. Despite that, the

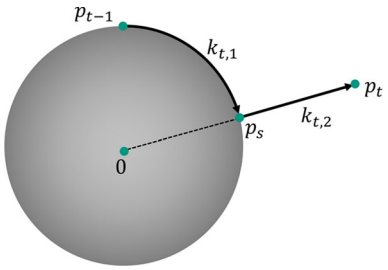

Fig. 3 Visualization of an approximation for the travel distance between two successive poses

agents' capability of minimizing travel distances given different modeling alternatives of state, action, and reward can be evaluated. The simplified approximation of the *travel distance* k_t ($t = 1, \ldots, T$) between two successive views is based on the poses p_{t-1} of the acquisition system of the previous interaction step $t-1$ and its pose p_t of the next interaction step t. This simplified approximation is visualized in figure 3. First, the *auxiliary point* p_s is determined. This point is located on the surface of the sphere, which also contains the point p_{t-1}, and is determined via the intersection of the vector to the point p_t starting from the origin. The approximated travel cost k_t is then composed of the *travel distance* $k_{t,1}$ from p_{t-1} to this intersection point p_s on the spherical surface and the *travel distance* $k_{t,2}$ from p_s to the pose p_t, which corresponds to the difference between the radius of the spherical surfaces on which p_{t-1} and p_t are located. If other than assumed and visualized in figure 3, the radius of p_{t-1} is greater than the one of p_t, the calculations of the travel distance is done the other way around. In the case of an equal radius, the travel distance solely consists of the distance $k_{t,1}$ on the spherical surface. The *cumulative travel costs* $k_{t,cum}$ as evaluation measure for an episode can then be calculated as the sum of the individual travel distances k_t between all interaction steps $t = 1, \ldots, T$ until the *terminating interaction step* T of the episode (see Eq. 7).

$$k_{t,cum} = \sum_{t=1}^{t=T} k_t \tag{7}$$

Implemented reward alternatives

Eight different dense reward variants have been implemented in this work and can be found in Table 3. The first variant is expressed by R_1, rewarding each acquisition step's relatively added object surface. A variation of this is modeled by R_2, where the agent is punished in case no additional object surface is acquired compared to the previous acquisition step. Rewarding the relative area to be newly covered, as in R_1 or R_2, leads to continuously diminishing rewards in later acquisition steps. This is because the object surface to be newly covered relative to the total surface area gets smaller, in case previous acquisition steps already provided a high coverage. Therefore, the reward signal R_3 considers the object surface covered in an acquisition step relative to the object surface that can still be acquired. R_4 additionally penalizes this reward modeling if no additional object surface is acquired. The reward variants R_5-R_8 expand the variants R_1-R_4 considered so far by integrating the travel costs of the robot k_t from acquisition step $t-1$ to t. To obtain R_5 - R_8, the rewards R_1-R_4 are scaled relative to the required travel distance k_t. This is intended to incentivize the agents to choose the next actions that lead to a high additional surface coverage at low travel costs in order to minimize the execution time of the entire view plan.

Similar approaches are adopted based on the agent's reward after an episode in a sparse reward setting. R_9 rewards the agent only on the object surface coverage reached after an episode. R_{10} additionally introduces a scaling concerning the number of acquisitions needed for the respective coverage achieved after an episode. Thus, the agent explicitly receives higher rewards for achieving a large object surface coverage while minimizing the number of acquisitions required. Instead of scaling by the number of necessary acquisitions, R_{11} scales by the cumulative travel costs $k_{t,cum}$. R_{12} then combines the two latter approaches.

The interplay between modeling the reward signal and defining an episode to influence the agent's behavior becomes evident using the reward signals R_{13} and R_{14}. The reward signal R_{13} rewards the agent after an episode solely based on the number of acquisitions performed. In this case, an episode's termination criterion is chosen so that an episode is only terminated after reaching a specified object coverage (e.g., 90 %). This ensures that the agent receives a high reward, provided it requires fewer acquisitions to reach the necessary surface coverage. An extension of this idea by integrating the travel paths is R_{14}. Thus, by combining the modeling of the reward signal and the definition of the termination criteria for an episode, the learning behavior and, thus, the performance indicators of the agent can be influenced in a targeted manner.

Agent modeling

The agent's configuration is varied in this work by the choice of strategy and the learning algorithm. For this purpose, the software framework stable-baselines (Raffin et al., 2021) is used, which supports the simple use and parameterization of different RL agents. Due to the continuous state and action space considered, *proximal policy optimization* (PPO) by Schulman et al. (2015) and *soft actor critic* (SAC) methods by Haarnoja et al. (2018) are used and evaluated. Both methods belong to the group of actor-critic variants of RL. These methods use a parameterized policy for action selection and a parameterized value function approximator. In figure 1, the *action policy parameters* for action selection are denoted

Table 3 Implemented modeling variants of the reward signal for the reinforcement learning agent

Abbreviation	Reward type	Formula
R_1	Dense	$r_t = \Delta COV_t = COV_t - COV_{t-1}$
R_2	Dense	$r_t = \begin{cases} \Delta COV_t = COV_t - COV_{t-1}, & \Delta COV_t > 0 \\ -1, & otherwise \end{cases}$
R_3	Dense	$r_t = \frac{\Delta COV_t}{COV_{rem}}$
R_4	Dense	$r_t = \begin{cases} \frac{\Delta COV_t}{COV_{rem}}, & \Delta COV_t > 0 \\ -1, & otherwise \end{cases}$
R_5	Dense	$r_t = \frac{\Delta COV_t}{k_t}$
R_6	Dense	$r_t = \frac{\Delta COV_t}{COV_{rem}} \cdot \frac{1}{k_t}$
R_7	Dense	$r_t = \begin{cases} \Delta COV_t \cdot \frac{1}{k_t}, & \Delta COV_t > 0 \\ -1, & otherwise \end{cases}$
R_8	Dense	$r_t = \begin{cases} \frac{\Delta COV_t}{COV_{rem}} \cdot \frac{1}{k_t}, & \Delta COV_t > 0 \\ -1, & otherwise \end{cases}$
R_9	Sparse	$r_t = \begin{cases} 0, & t < T \\ COV_t, & t = T \end{cases}$
R_{10}	Sparse	$r_t = \begin{cases} 0, & t < T \\ \frac{COV_t}{t}, & t = T \end{cases}$
R_{11}	Sparse	$r_t = \begin{cases} 0, & t < T \\ \frac{COV_t}{k_{t,cum}}, & t = T \end{cases}$
R_{12}	Sparse	$r_t = \begin{cases} 0, & t < T \\ \frac{1}{t} \cdot \frac{COV_t}{k_{t,cum}}, & t = T \end{cases}$
R_{13}	Sparse	$r_t = \begin{cases} 0, & t < T \\ \frac{1}{t}, & t = T \end{cases}$
R_{14}	Sparse	$r_t = \begin{cases} 0, & t < T \\ \frac{1}{t} \cdot \frac{1}{k_{t,cum}}, & t = T \end{cases}$

by w_ψ, and the *value function parameters* are denoted by w_ϕ. During the training phase, weights w_ϕ are continuously updated to approximate the value function given the current state. This, in return, enables the evaluation and optimization of the action selection strategy by updating its weights w_ψ by gradient-based updates. The representation of the strategy or the estimators of the value function is carried out, as usual in the current state of research, using neural networks (c.f. Devrim Kaba et al. (2017); Potapova et al. (2020); Korbach et al. (2021); Landgraf et al. (2021); Mnih et al. (2015)). For the use case of this work, any network architecture that transfers a point cloud as input into a feature vector can be used. Based on preliminary tests, two suitable alternatives for processing point clouds have been identified. First, an implementation of PointNet by Qi et al. (2017) is used, the first implementation of a neural network for learning on point clouds. In the PointNet Encoder, the input point cloud is transformed and processed by a series of input transformation layers and multi-layer perceptrons (MLPs) whose weights are shared across all points. The input transformation layers take points or features and apply pose normalization by multiplying the input by a transformation matrix, which is output by a subnetwork called T-net. The shared MLP processes each feature of this transformed feature vector independently to learn local feature representations. The output of the MLPs is then aggregated using a max-pooling layer to obtain a fixed-length global feature vector that captures information about the entire point cloud.

The second option considered is the use of the feature vector as the output of the encoder of the *Point Completion Network* (PCN) by Yuan et al. (2018), which builds on the approach of the PointNet. A shared MLP calculates local point features. Using a max-pooling layer, intermediary global features are obtained and concatenated with the local point features. Using another shared MLP and max-pooling, the final global feature vector is obtained. In this work, a regression MLP is added to the respective encoders of PointNet and PCN to create the complete neural networks used. The output dimension n of the regression MLP corresponds to the number of free parameters of the action modeling. For details on the hyperparameters used and the network architectures, please refer to Section "Agent configuration and parameters" as well as the original works of Qi et al. (2017) and Yuan et al. (2018). An overview of the network structure when using the PCN encoder or PointNet encoder can be found in Fig. 4.

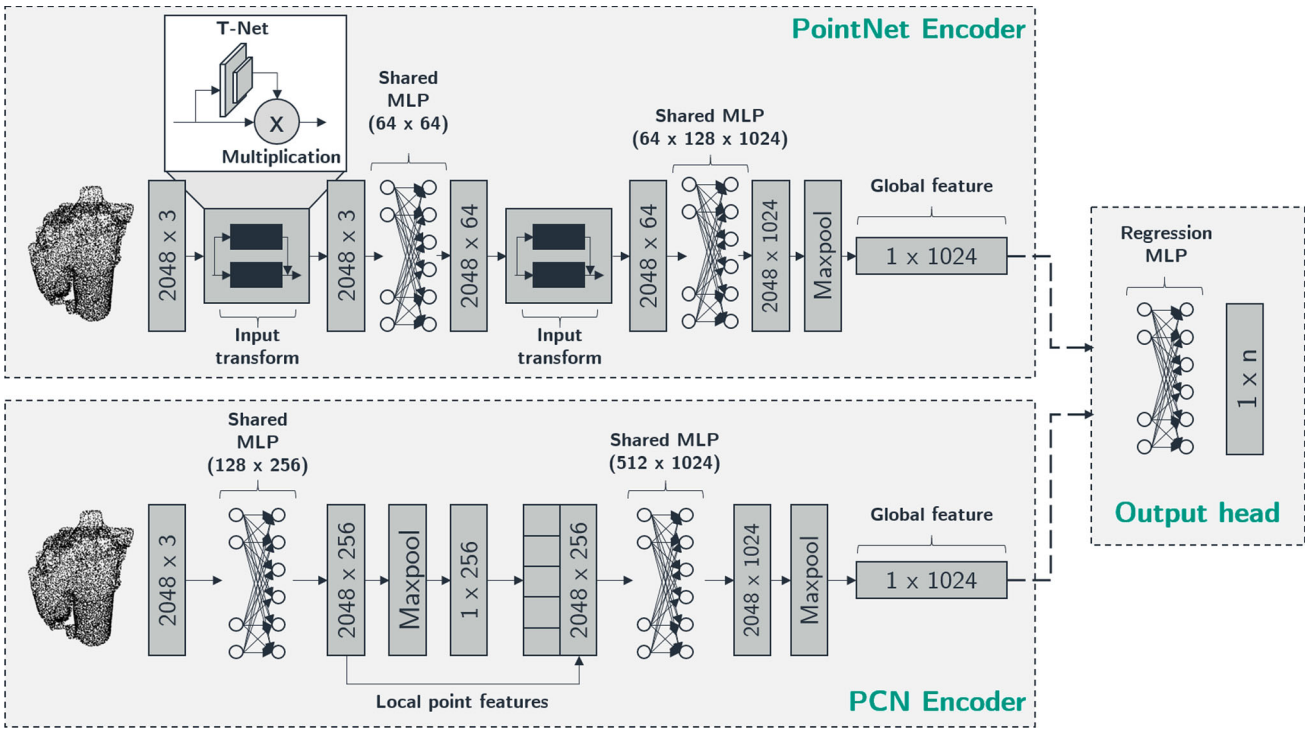

Fig. 4 Visualization of the implemented network structures in this work based on the PCN by Yuan et al. (2018) and the PointNet by Qi et al. (2017). Note that the input in the visualization is $s_{t,cov}$ with size 2048 × 3. When using $s_{t,bin}$ it would be 2048 × 4

Use case description and computational results

To simulate the inspection process in remanufacturing, a dataset of synthetically generated starter engines that represent a product being remanufactured in reality is used in the present work. By varying as few as possible parameters of the RL agent, we can explore the influences of the varied parameters on the agent's performance. The general learning ability for the NBV problem is shown first, using the achieved object coverage as a performance indicator. We then compare the previously described alternatives in state modeling, learning algorithm, point cloud encoder, action modeling, and reward modeling alternatives. We thereby do not focus on finding the best modeling alternative, but rather an identification of different modeling alternatives on the agent behavior. Furthermore, training and testing of the best-performing agents is conducted with real starter motor models and compared to benchmark algorithms.

Dataset used for comparing modeling alternatives

The present work makes use of a dataset of starter engines that is automatically generated. By this, the properties of the remanufacturing use case under consideration are addressed. The objects for which the RL-agent generates poses of the acquisition system vary in their geometric properties. This is necessary since many different starter engine variants exist from different manufacturers. In addition, the engines may be damaged or even missing entire parts, resulting in significant differences. The approach is based on an existing pipeline approach from Wu et al. (2022), adapted for generating starter engines.

The starter engines are created randomly from nine components (e.g., solenoid, gear housing, connector, flange), which are varied by 28 different parameters (e.g., height or diameter). An illustration can be seen in figure 5. The parameters are kept within a realistic range by parameter limits and clearly defined relationships to each other. This ensures a diverse and realistic data set. As a data basis for the present work, 100 engines with random parameters were generated. The engines are generated and saved in STL format for training the agents.

Training procedure and theoretical real-world application

The agent training is performed in episodes. Each episode represents the acquisition process of one generated starter engine. To simulate randomness in positioning, the STL model of a randomly selected engine is loaded upright into the simulation and rotated around its longitudinal axis (z-axis) in the range of $[-180°, 180°]$. Furthermore, it is tilted in the $[-25°, 25°]$ range in either the x or y-axis. The acqui-

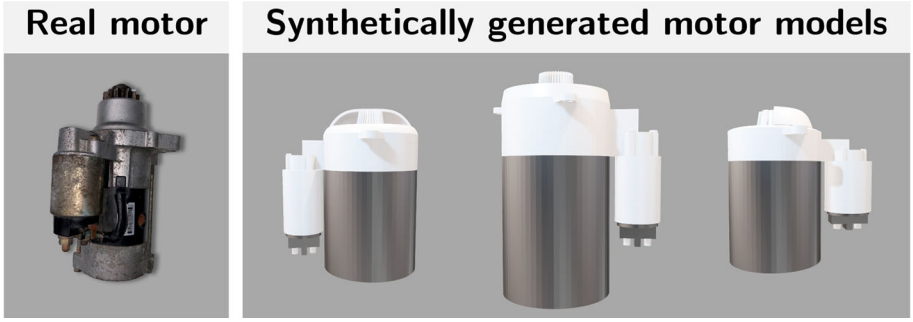

Fig. 5 Visualization of a real engine and three variants of generated starter engines with varying generation parameters

sition process is then performed until the terminal state s_T is reached, either by reaching 90% surface coverage or the threshold of ten acquisitions, depending on the episode design. The state, action, and reward transitions are used to optimize the strategy π of the agent to select actions.

It has to be noted that despite calling our approach model-free, in order to deduce the performance indicators during training and testing as well as to calculate the reward based on these performance indicators during training, a model of the object needs to be present. This applies to agents using states $s_{t,cov}$ as well as $s_{t,bin}$. However, when using $s_{t,cov}$, our approach is model-free during application. Just as in supervised learning, where ground truth data exist during the training of the model, no object model or ground truth data are available during the application (when using state $s_{t,cov}$). This is due to the fact that in a real application scenario, the object is available to derive the states. In this case, the state $s_{t,cov}$ is solely deduced based on information that can be extracted using point clouds of previous acquisitions, i.e., information that does not rely on an object model. Ideally, in an application scenario, a reward signal is not needed since a trained agent is used. Only finetuning in the sense of a Sim-2-Real transfer, where one or more real object models are needed, is a possibility. Since, in this case, the actual object coverage can no longer be calculated, the heuristic determination of a fixed number of acquisitions (e.g., 4) is necessary. Alternatively, an agent can be trained, which, in addition to the next pose of the acquisition system, outputs whether the episode should be aborted. However, we do not consider the implementation and evaluation of such an agent in this paper and leave it open for future research.

Agent configuration and parameters

Unless specified otherwise, the parameters seen in Table 4 of the simulation are assumed to be fixed for the evaluation of different modeling alternatives of the agent. In the case of the parameters for the sensor model, they have been derived from the specifications of a real three-dimensional acquisition system.

We chose a required object coverage of 90% for the agent to achieve to account for self-occlusions or other phenomena that would prevent achieving 100% object coverage for certain object models. Even though reaching the required object coverage of 90% can be achieved with fewer acquisitions, a maximum number of acquisitions in an episode of 10 is allowed to enable the agent to explore. In addition, a criterion for terminating a learning cycle is introduced. Stable-baselines define this by the maximum number of agent interactions with the environment. This maximum number of interactions is set to 75.000 based on preliminary tests. After this number of interactions, it is predictable whether an agent can learn, and if so, all considered agents have converged.

The learning rate and the discount factor were also determined based on these preliminary tests. All agents considered in the preliminary tests showed stable and repeatable learning behaviors for these parameters, which is necessary for quantitative comparisons of the modeling alternatives of the agents. As the learning rate has a low value of 0.000078 with linear decay and discount factor, a value of 0.9 is used to ensure convergence.

As a standard agent configuration, we use the SAC algorithm as introduced in Section "Agent modeling". The networks for approximating the policy and the value function consist of a PCN encoder and a downstream regression MLP with two layers of sizes 128 and 64 and the output layer. Preliminary evaluations determined the layer structure to show promising convergence behavior for arbitrary agent configurations. Unless specified otherwise, $s_{t,cov}$, A_{3S2R_lim} and R_2 are used as state action and reward. For the mapping of radius, r using spherical coordinates and x, y, z using Cartesian coordinates, the mapping without using prior knowledge $M_{npk}^{x,y,z}$ was used. In contrast, the action definition A_{3S2R_lim} explicitly uses prior knowledge concerning the Euler angles α and β ($M_{wpk}^{\alpha,\beta}$. The standard configurations of the agent were chosen in such a way that for variation of individual factors (e.g. reward, state, etc.) convergence behavior continues to occur and the variation of these individual factors can then be evaluated quantitatively and qualitatively, thus allowing an interpretation of the results.

Table 4 Default agent configuration parameters

Sensor model parameters	Values
Near and far bounds	$[30, 50]cm$
Resolution	$(430 \times 300)_{xy}$
Opening angles	$(27°, 25°)_{xy}$
Learning cycle design	**Values**
Required object coverage	90%
Maximum number of acquisitions	10
Maximum number of steps	75.000
Learning parameters	**Values**
Learning rate	0.000078^a
Discount factor γ	0.9
Agent design	**Values**
Algorithm	SAC
Encoder	PCN
Regression MLP	128 x 64
State	$s_{t,cov}$
Action	A_{3S2R_lim}
Mapping	M^{r}_{npk}, $M^{\Delta\alpha,\Delta\beta}_{wpk}$
Reward	R_2

^{a} denotes a linear decay with a final value of 0

Evaluated performance indicators

In order to compare the performance of different modeling alternatives, performance indicators are used in this work. Figure 6 shows two exemplary courses of the performance indicators of the agents during a learning cycle. These performance indicators have already been derived in Section "Reward module" and are:

- Coverage
- Number of acquisitions needed to end an episode
- Travel distance

Because of randomness in the behavior of the RL-Agent, multiple simulation runs with the same configuration are performed. Using three runs per configuration results in an adequate confidence interval in the agent's converged state, allowing quantitative comparisons.

The performance indicators in Fig. 6 are plotted over the number of episodes. One episode encompasses the acquisition process of one object model by the agent. Since multiple agents were trained for each agent configuration, the plots in Fig. 6 contain the average trajectory of the performance indicator in a bold line. Additionally the minimum and maximum value of all agents trained for each episode in gray.

In addition to the performance indicators, the course of the rewards received by the agent over the episode is also shown. The course of the reward signal indicates whether an agent is, in principle, able to learn with a given reward signal and thus increases or maximizes the reward it receives. By comparing the course of the rewards received by the agent with the performance indicators of interest over the learning time, it is possible to determine whether the reward signal was modeled correctly. If necessary, it is also possible to iteratively adjust the modeling of the reward signal.

As shown in Fig. 6, the agents reach a converged state after a certain number of episodes. In the converged state, no significant improvement in performance indicators is identifiable until the final value of 12,000 episodes is reached. For each modeling alternative, all data to reconstruct these plots were saved. For simplicity, in the remainder of this work, the modeling variants are compared on the basis of the performance indicators in the converged state. The numerical value in the converged state is calculated by averaging the respective performance indicator over the last 100 episodes of a simulation run and over the simulation runs carried out with this respective modeling alternative.

Computational and evaluation results

In this section, we first show the general learning capability of the agent using the NBV problem. Then, the influences of the different modeling alternatives of state modeling, learning algorithm, point cloud encoder, action modeling, and reward modeling on the performance indicators are investigated. Since the training of agents is time and computationally intensive, we only follow a partially exhaustive approach where every combination is examined concerning the modeling alternatives. Preliminary results indicated the suitability of first examining the parameters with fewer options (state, algorithm encoder) and later the parameters with more options (action and reward). In each case, the design option that shows the best performance continues to be pursued. Our experiments are conducted on a Linux system with an AMD Ryzen Threadripper PRO 3955WX with 16 cores running at 3.9 GHz and NVIDIA RTX A6000 graphics. The average runtime of each experiment is approximately 10 h.

Verification of agent learnability for the NBV problem

Question: *Is it possible to train learning agents with the modeling alternatives presented and how well do these agents perform in relation to random benchmarks?*

To obtain a baseline for verifying the learning ability of RL agents, the performance indicators of two agents that choose actions randomly are first determined for the most straightforward problem class, the NBV problem. The framework generates an initial fixed pose of the acquisition system, as stated in Table 5, to let the random agents return an action corresponding to the NBV. The first random baseline agent

NBV_1^{rand} makes use of the action variant A_{3S2R}, here in use with a mapping on a [0, 100] radius interval (M_{npk}^r for radius r), with the maximum number of degrees of freedom possible. The attachment $rand$ points out the property of a randomly created action vector. NBV_1^{rand} shows no learning behavior regarding all the performance indicators - a trivial result given random action selection. No additional coverage compared to the initial acquisition is achieved. All the total object coverage can be associated with the initial acquisition, which usually accounts for around 44% of the object surface. Reducing the degrees of freedom from 5 to 3 using the A_{3S0R} action modeling with prior knowledge regarding the orientation toward the objects' center ($M_{wpk}^{\alpha,\beta}$) results in the second random baseline agent NBV_2^{rand}. In this case, the acquisition system is automatically orientated to the object's center of gravity, and the radius is mapped on the operating range of [30, 50] (M_{wpk}^r for radius r). Additional coverage can be obtained with said random baseline. Still, the action vector is created randomly in this case, showing that integrating prior knowledge improves the performance, although the agent does not learn anything.

While the action vectors of NBV_1^{rand} and NBV_2^{rand} are created randomly, two further agents NBV_3 and NBV_4 are trained, which choose their actions based on a learned policy (see Table 4 for parameters). Their actions are the results of processing the point cloud of the initial acquisition in the PCN encoder and the downstream MLP. In both cases, the A_{3S0R} action with automatic focus on the object's center of gravity is used. NBV_4 only differs from NBV_3 in the pose of the acquisition system for the initial acquisition, which is generated randomly for NBV_4. Table 5 shows a significant coverage improvement of approximately 30% for agents NBV_3 and NBV_4 compared to NBV_2^{rand}. This shows that the agents are generally capable of learning. Additionally, the non-existing differences in final surface coverage between NBV_3 and NBV_4 show the ability of NBV_4 to reactively choose the pose of the subsequent acquisition according to the varying pose of the initial acquisition.

In addition to the surface coverage, the number of acquisitions and the travel distances are shown in Table 5. Since all agents use the reward signal from the default agent configuration in Table 4, the reward primarily maximizes the object coverage. A comparison of lengths of travel paths as well as the number of acquisitions, which is always two, is therefore not relevant and is omitted in the context of comparing random baseline agents to RL agents.

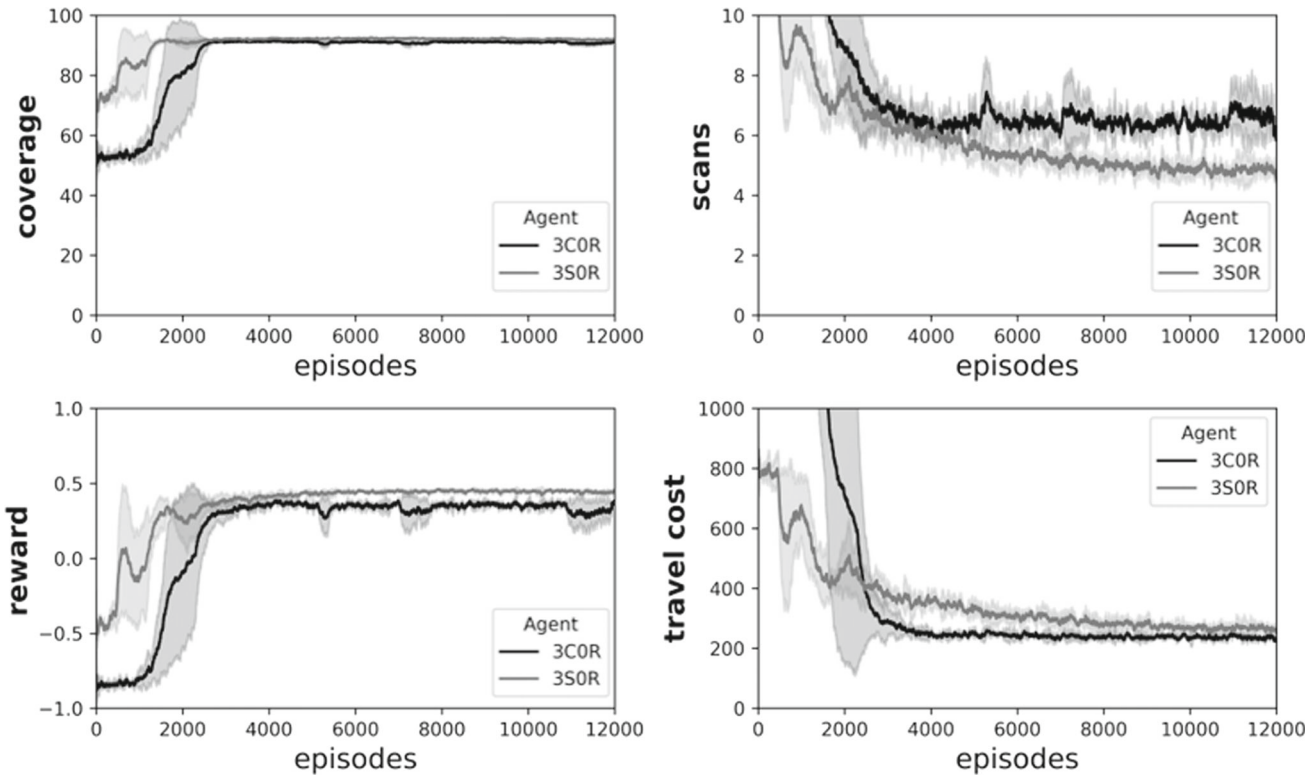

Fig. 6 Trajectory of evaluated performance indicators and reward of two exemplary agent configurations over the course of learning. Top left: Trajectory of the object surface coverage. Top right: Trajectory of the required acquisitions (scans) to reach 90 % surface coverage. Bottom left: Trajectory of the reward signal. Bottom right: Trajectory of cumulated travel costs

Table 5 Results for Baseline RL-agents with varying action encoding and position of initial acquisition

Agent	Action	initial	Mapping	Coverage[a]	Coverage[b]	Acquisitions	Travel cost
NBV_1^{rand}	A_{3S2R}	fix	$M_{npk}^{\alpha,\beta}$, M_{npk}^{r}	0.00	44.24	2.00	82.00
NBV_2^{rand}	A_{3S0R}	fix	$M_{wpk}^{\alpha,\beta}$, M_{npk}^{r}	9.28	53.35	2.00	80.74
NBV_3	A_{3S0R}	fix	$M_{wpk}^{\alpha,\beta}$, M_{npk}^{r}	39.42	83.07	2.00	120.06
NBV_4	A_{3S0R}	random	$M_{wpk}^{\alpha,\beta}$, M_{npk}^{r}	**39.62**	**83.23**	2.00	122.63

Bold values indicate best achieved
[a]: Coverage of initial acquisition is excluded
[b]: Coverage of initial acquisition is included
^{rand}: Actions are calculated randomly thus, the agent is not capable of learning

Table 6 Results for RL agents with changing state encoding, learning algorithm, and point cloud encoder network variant

Agent	State	Algorithm	Encoder	Coverage	Acquisitions	Travel cost	Steps to convergence
1	$s_{t,cov}$	SAC	PCN	92.01	5.69	305.55	28455
2	$s_{t,bin}$	SAC	PCN	**92.05**	5.70	**293.69**	**12550**
3	$s_{t,cov}$	PPO	PCN	91.82	6.96	429.43	52333
4	$s_{t,cov}$	SAC	PointNet	91.88	**5.38**	304.53	33667

Bold values indicate best achieved

Answer: *By comparing agents that learned to perform a NBV with benchmark agents that randomly chose actions, the learnability of the agents can be verified.*

Comparison of state modeling alternatives

Question: *Which of the introduced state encoding alternatives, $s_{t,cov}$ or $s_{t,bin}$,performs better?*

Two state encoding alternatives are introduced in Section "State module". First, the state can be based on and calculated by the acquired point cloud $s_{t,cov}$ up until the current interaction step t (model-free). Second, the binary state encoding passes a point cloud $s_{t,bin}$ based on and calculated by the object model with information on whether a point has already been seen to the agent(model-based). These two variants are compared in this section.

Agent 1 and 2 in Table 6 display the results for the state alternatives $s_{t,cov}$ and $s_{t,bin}$ considered. On average, both agents can reach the required object coverage criteria of 90% surface coverage in the last 100 episodes. Regarding the acquisitions, both agents lower the needed number of acquisitions from 10 to less than 6 on average. This can be attributed to the discount factor $\gamma = 0.9$, which encourages the agent to perform acquisitions with a high reward at earlier steps. Thus, the desired surface coverage of 90% is achieved in fewer steps (acquisitions), reducing the number of acquisitions required. Additionally, as the number of acquisitions increases, it becomes more difficult for the agent to find and acquire unknown regions, increasing the probability of penalty (based on the reward R_2 used). Considering the performance indicators obtainable, there are no significant differences between the introduced state encodings. By comparing the needed steps of the agents to converge, it can be stated that a faster convergence of Agent 2 using $s_{t,bin}$ is achieved compared to Agent 1 using $s_{t,cov}$. This can be attributed to the more complete information content passed to the agent. In the case of binary coding $s_{t,bin}$, the agent receives information about all possible points to be captured and those that have been captured so far and are yet to be captured. The latter information is missing in the case of the first state alternative $s_{t,cov}$, where the agent only receives information about points captured so far.

Thus, by using the proposed framework, agents can optimize performance indicators given a suitable combination of state action and reward modeling. This is either possible by a state encoding alternative where a model of the object is needed ($s_{t,bin}$) but also with an alternative that does not explicitly need a model ($s_{t,cov}$). Although there could be settings where a binary encoding is valuable or even necessary in order to reach a performant agent, by using the state encoding $s_{t,cov}$, it can be made sure to only use the information available in a real industrial use case, such as remanufacturing, in which no object model might exist at runtime. Due to these findings, in the following agent setups, the state encoding $s_{t,cov}$ using the acquired point cloud until interaction step t is used in further evaluations (see Table 4).

Answer: *Both state encoding alternatives, $s_{t,cov}$ or $s_{t,bin}$,are performing equally well regarding the view planning task.*

Table 7 Results for RL-agents with changing action variants

Agent	Action	Mapping	Coverage	Acquisitions	Travel cost
Action output in cartesian coordinates					
5	A_{3C0R}	$M^{\alpha,\beta}_{wpk}$, $M^{x,y,z}_{npk}$	90.97	6.58	**236.30**
6	A_{3C2R}	$M^{\alpha,\beta}_{npk}$, $M^{x,y,z}_{npk}$	43.49	11.00	1556.21
Action output in spherical coordinates					
7	A_{3S2R}	$M^{\alpha,\beta}_{npk}$, M^{r}_{npk}	44.19	11.00	838.16
8	A_{2S0R}	$M^{\alpha,\beta}_{wpk}$, M^{r}_{wpk}	**92.20**	5.26	296.02
9	A_{3S0R}	$M^{\alpha,\beta}_{wpk}$, M^{r}_{npk}	92.16	**4.63**	258.08
10	A_{2S2R_lim}	$M^{\Delta\alpha,\Delta\beta}_{wpk}$, M^{r}_{wpk}	91.71	6.25	296.99

Bold values indicate best achieved

Comparison of learning algorithms

Question: *Which of the introduced learning algorithms, PPO or SAC, performs best?*

Since continuous state and action spaces are considered in this work, two different state-of-the-art learning algorithms (PPO and SAC) for this problem set are evaluated. Agent 1 uses the off-policy SAC algorithm, while Agent 3 uses the on-policy PPO algorithm. Performance indicators are depicted in Table 6.

Both agents are able to optimize the object coverage up to around 90% and lower the needed acquisition steps to less than 7 acquisitions. Despite this, Agent 1 trained with the SAC algorithm outperforms Agent 3 using the PPO algorithm in nearly all performance indicators (object surface coverage, number of acquisitions, and cumulated travel costs). Besides, Agent 3 using PPO exhibits poor convergence behavior, reflected in the number of steps required to converge. These findings are consistent with the claims of Haarnoja et al. (2018) that on-policy algorithms, such as PPO, have a lower sample efficiency compared to off-policy algorithms, such as SAC, and thus require more steps for convergence. In addition, a qualitative result is that with the same hyperparameterization when using the PPO, divergence and thus no convergence of the agent occurs more often after initial learning behavior. In contrast, a learning run of a SAC agent almost always leads to convergence. For the use case considered in this work, the SAC algorithm proves to be preferable due to its more stable learning behavior. Due to better convergence behavior and the SAC outperforming the PPO concerning the performance indicators, the SAC algorithm is chosen for further experiments (see also 4).

Answer: *The agents using SAC perform better compared to agents trained with PPO. Additionally, agents using SAC show more robust convergence behavior.*

Influence of used point cloud encoder

Question: *Which of the introduced point cloud encoders, PCN or PointNet, performs best?*

This work considers and evaluates two variants to extract point clouds' features concerning the input's permutation invariance. While Agents 1–3 use the PCN, Agent 4 uses the PointNet. Comparing the results of Agent 1 to Agent 4 shows that both agents can optimize the object coverage to more than 90% and lower the necessary amount of acquisitions to less than six while showing stable convergence behavior. Using the PointNet encoder results in a three-fold increase in training times. This is due to the increased computation time based on the increased amount of weights using the PointNet (3104294 weights) compared to the PCN (1126980 weights). This is also why we omitted to implement more sophisticated networks working with point clouds since these tend to have more parameters and/or additional intermediate calculation steps (for example, sampling, grouping, and ball querying in PointNet++), which additionally increase calculation time. Due to this, for further experiments and agent settings, the PCN is used to process input point clouds as depicted in Table 4. However, in future works, we plan to adopt such sophisticated networks.

Answer: *PointNet performs comparably concerning the performance indicators but needs three times the inference time than PCN*

Comparison of action encodings

Question: *Which of the introduced action encodings performs best?*

Considering the action encodings, there are two different possibilities for the output of the pose by the agents. An action can be mapped to Cartesian coordinates or to spherical coordinates. In addition, different action alternatives result from integrating prior knowledge. The results of the evaluation of the different action encodings are depicted in Table 7.

Agents 5 and 6 are trained by mapping the agents' actions to Cartesian coordinates. While Agent 5 makes use of the automatic orientation of the acquisition system to the center of the object, Agent 6 also needs to output values to determine the orientation of the acquisition system. Considering both agents, significant differences become evident. While Agent 5 can increase the achieved object coverage to more than 90% and lower the acquisitions as well, Agent 6 cannot optimize any performance indicator. Using the maximum possible number of 10 acquisitions after the initial one, the agent covers no object surface apart from the one achieved with the initial acquisition. Thus, the agent cannot learn to output actions that allow the orientation towards the object's center, which indicates that the object is never in the acquisition system's field of view. Problem-solving the VPP by the agent without further restrictions (Agent 6) is therefore impossible in this case, and restrictions (e.g., automatic orientation of the acquisition system to the object center, Agent 5) must be introduced.

Similar results can be obtained considering Agents 7 to 10 using actions mapped to spherical coordinates. Agent 7, which again has to solve the VPP without further restrictions, does not learn a strategy to optimize performance indicators. This makes the significant complexity of correct acquisition system rotation towards the object evident. Again, Agent 8 and Agent 9, which use the automatic orientation to the object's center, exhibit favorable behavior. The only difference between the two agents is that Agent 9 has to learn the optimal distance r between the acquisition system and the object center, whereas this is predefined for Agent 8 ($r = 47$). Both agents optimize the achieved coverage to more than the required object coverage of 90%, with Agent 8 being slightly better regarding object coverage and the required number of acquisitions. In-depth analysis have shown that although Agent 9 performs many empty acquisitions in its first episodes, Agent 9 can quickly shift the action vector component responsible for determining the radius toward the permissible near and far bounds of the frustum. As in the Cartesian coordinates case, Agents 7–9 can correctly place themselves in space regarding these restrictions.

The last agent evaluated in this series of experiments is Agent 10, which can specify an angle-based deviation from the orientation of the acquisition system to the object center point through its action vector ($M_{wpk}^{\Delta\alpha,\Delta\beta}$). Agent 10 thus exploits the prior knowledge of the orientation towards the object center but can readjust by additionally outputting an angle-based correction. Thus, a better strategy may be learned by selectively exploiting different orientations of the acquisition system. The results show that a learning effect also occurs in this case, but the agent performs worse than agents 8 and 9 concerning the performance indicators. Comparing Agent 10 with Agent 1, they only differ in their prior knowledge of the radius r. Agent 10 is always positioned at a fixed distance (47 cm) from the center of the object (M_{wpk}^{r}), while Agent 1 is able to position itself freely by learning the distance to the inspection object (M_{npk}^{r}). Interestingly, Agent 1 performs better than Agent 10 in the performance indicators object coverage and needed number of acquisitions. Detailed analyses have shown that this is because Agent 1, on average, moves to positions with a distance of about 49 cm from the object center. This positioning further away from the inspection object results in a larger object surface that can be covered with one acquisition, making fewer acquisitions necessary but resulting in higher travel costs.

Comparing agents using Cartesian and spherical action mappings with the same degrees of freedom (Agents 5 and 9), the one using the spherical coordinate system (Agent 9) outperforms its counterpart. One possible explanation is that in the case of spherical coordinates, the radius can be optimized by only one action vector entry. In contrast, in the case of Cartesian coordinates, the distance to the origin results from three entries (x, y, z-coordinates).

In general, increasing degrees of freedom by adding orientation lowers the performance. Only Agents using the action mapping A_{3S2R_lim} or A_{2S2R_lim} integrating prior knowledge $M_{wpk}^{\Delta\alpha,\Delta\beta}$, are able to reach the coverage criteria when being tasked with determining the acquisition systems' orientation. The agent can slightly readjust the orientation of the acquisition system based on prior knowledge of the automatic orientation used. Thus, this action modeling is used in the following experiments, even though it performs slightly worse than Agents 8 and 9. This is because different modeling alternatives of the RL agents may use said degree of freedom in the action space in conjunction with a suitable reward signal to find optimal strategies.

Answer: *Mapping to spherical coordinates outperforms mapping to Cartesian coordinates. In general, the agents are not able to learn the adjustment of the orientation of the agent and prior knowledge has to be incorporated for the agent to perform well on the acquisition planning task.*

Comparison of reward alternatives

Question: *Which of the introduced reward alternatives performs best?*

This section details the experiments concerning the reward alternatives found in Table 3. Two approaches are evaluated. First, the use of dense rewards that evaluate the agent's action after each acquisition (see agents 11 to 18 in Table 8). Second, sparse rewards evaluate the quality of multiple actions of the agent after the completion of a whole episode (see agents 19 to 24 in Table 8).

Agent 11, using a straightforward dense reward R_1 calculated by the additionally achieved coverage of one acquisition, performs poorly regarding the performance indicators. Needing ten acquisitions on average to reach 90% object sur-

Table 8 Results for RL agents with changing rewards

Agent	Reward	Coverage	Acquisitions	Travel cost
Dense rewards				
11	R_1	84.66	10.00	788.99
12	R_2	**92.09**	6.01	296.55
13	R_3	91.62	6.55	310.14
14	R_4	91.18	6.07	257.54
15	R_5	84.28	10.48	**208.13**
16	R_6	66.58	10.78	859.97
17	R_7	89.49	7.94	269.83
18	R_8	91.91	**5.24**	249.68
Sparse rewards				
19	R_9	67.76	10.86	414.01
20	R_{10}	**91.06**	**6.16**	**251.31**
21	R_{11}	65.91	10.88	855.25
22	R_{12}	65.38	10.91	866.02
23	R_{13}	67.72	10.74	832.44
24	R_{14}	67.07	10.90	845.95

Bold values indicate best achieved

face coverage shows that by applying R_1, the coverage may be optimized, but the agent is not incentivized to reduce the number of acquisitions of an episode.

Agent 12, using R_2, is therefore penalized when an acquisition is taken that does not acquire additional surface points of the object. This results in significant surface coverage gain while the number of acquisitions is reduced simultaneously. Because of this and the simple nature of R_2, we chose this reward signal for all previous evaluations as the default reward signal (compare Table 4).

Agent 13 and 14 are rewarded in a different way. The reward is calculated by scaling the object surface acquired in the previous acquisition in relation to the maximum acquirable surface possible. Other than simply using object coverage (Agent 11 with R_1), Agent 13 can reach the object surface criteria of 90% on average without being penalized when performing acquisitions with no additional surface coverage. The reward variant of Agent 14 makes use of the penalties for empty acquisitions, which does not significantly increase the coverage.

Agents 15 and 16 are variants of Agents 11 and 13. The used reward signals R_5 and R_6 additionally introduce scaling terms for the travel cost in the reward signals of Agents 15 and 16 (as explained in Section "Approximation of the length of traveling distance") to reduce the cumulated travel paths between subsequent acquisitions. However, introducing additional terms into the reward calculation significantly worsenes the performance regarding the performance indicators object surface coverage and number of acquisitions. Both agents cannot reach the required surface coverage criteria of 90% to terminate an episode, even though Agent 15 still reaches a higher surface coverage than Agent 16. Regarding the travel cost, agent 15 achieves a comparatively small and thus good result. Agent 16 exhibits one of the highest reported travel costs in all experiments. This result is a hint that modeling a reward signal that considers multiple performance indicators does not always lead to the optimization of all performance indicators, even though they are integrated into the calculation of the reward. It can be concluded that a combination of multiple performance indicators in one reward signal has a negative effect on the overall result. A single reward value potentially resulting from a plurality of influencing factors (newly gained surface coverage, travel costs, etc.) may be too complex for the agent to learn.

Although Agents 15 and 16 are not performing well, we also evaluate if a penalizing term for empty acquisitions has a positive effect, as it has, e.g., with Agent 12 compared to Agent 11. Agents 17 and 18 result from the rewards R_7 and R_8 and are refining Agents 15 and 16 with R_5 and R_6 by penalizing the agent if empty acquisitions are made. Both agents 17 and 18 reach a better surface coverage than their counterparts, Agents 15 and 16, by obtaining values of nearly or more than 90%. Besides, they are also able to lower the number of acquisitions needed. Considering the results of previous rewards, the beneficial effect of using penalties for empty acquisitions through placement out of the permissible near and far bounds of the frustum or wrong orientation becomes evident.

The following Agents, 19–24, are configured with rewards that are not given after every action from the agent but after every finished episode. Solely using the achieved surface coverage after the end of an episode, Agent 19, using reward R_9, cannot optimize any performance indicator. Different from all other agents with sparse rewards, Agent 20 converges. With the reward R_{10} combining both the surface coverage of an episode and the number of acquisitions required, an average coverage exceeding the required one can be obtained. Besides, the number of acquisitions is reduced. Different from Agent 20, the reward R_{11} of Agent 21 is calculated from the obtained object coverage and the combined travel cost of an episode. Here, neither coverage nor the required acquisitions could be optimized. Furthermore, the travel costs are still exceptionally high, even though they are included in calculating the reward. This is also the case for Agent 22, whose reward R_{12} is a combination of the rewards R_{10} and R_{11}.

Other than all previous agents and rewards, an episode of Agents 23 and 24 solely ends after the object surface coverage criteria of 90% is reached and not after a maximum number of acquisitions of 10 acquisitions performed by the agent (maximum permissible number of acquisitions is 11 including the initial acquisition). The reward is calculated only by the number of acquisitions taken in one episode (R_{13}) or

by the number of acquisitions and the associated travel cost (R_{14}). In both cases, the Agents cannot optimize any performance indicator regarding the coverage of less than 70% and the mean number of needed acquisitions of more than 10. These observations show that it is possible but challenging, using sparse rewards, to train agents for the problem at hand.

Answer: *Dense rewards have shown to outperform sparse rewards. Penalizing the agents for performing empty acquisitions always leads to improvement of the learned strategy. Optimization of multiple performance indicators through combination into a single reward signal has not shown positive results and may be subject to future work.*

Comparison with benchmark algorithms

Question: *How do the best-trained agents perform in comparison to benchmark algorithms?*

The analysis results of the different parameters and modeling alternatives have shown that agents can be trained to learn and solve the VPP. These agents can be compared with traditional benchmark algorithms. The results are in Table 9. Since optimizing the travel paths proves challenging for the agents, it is not included in this comparison. The heuristic H_{rec} and two variants of an analytic algorithm B_{SCP} are used for comparison. The heuristic H_{rec} represents a human-like solution of how an acquisition system can be positioned around an approximately rectangular object. The analytical solution method B_{SCP} explicitly takes into account the geometry model (stl) and uses this model to plan the optimal poses of the acquisition system.

The first benchmark heuristic H_{rec} represents a simple solution by placing a rectangular bounding box around the given mesh of the object and generating a pose of the acquisition system for each face of the bounding box. A pose is generated starting from the surface center of the respective side of the bounding box. For this purpose, a point is generated in the direction of the surface normal of the side surface at the optimal working distance of the acquisition system. The orientation of the acquisition system is then determined to point to the object's center. Since, in this case, three-dimensional Cartesian coordinates are used to define the position of the acquisition systems' pose and the rotation is fixed, this corresponds to the action mapping A_{3C0R}. For each determined pose, an acquisition is performed, and a surface coverage percentage is calculated. The results in Table 9 then correspond to the average coverage for all available models.

The second benchmark algorithm B_{SCP} is an analytical solution method for the VPP. Given the object mesh to be inspected, specific positions are equally sampled on a spherical surface centered in the object center. The radius is chosen as the optimal working distance of the acquisition system. For this work, 900 positions are sampled. The orientation is then, as with the first benchmark heuristic H_{rec} set, so the acquisition system points toward the object center. Hence, this corresponds to the action mapping A_{2S0R}. All 900 poses are evaluated for each model, and the points on the ground truth point cloud P_{GT} that can be acquired with said pose are stored. Then, an algorithm is used to determine the n poses of the 900 available poses that obtain the highest amount of surface coverage. In an iterative manner, this algorithm loops over the results of 900 acquisitions. In each iteration, it selects the pose that acquires the most amount of surface points on P_{GT} that have not been acquired with the previously selected poses. This algorithm essentially solves the so-called *Set-Covering-Problem* (SCP) the VPP can be framed as. The parameter n is a hyperparameter and therefore, two variants of the analytic solution method $B_{SCP,4}$ and $B_{SCP,5}$ with $n = 4$ and $n = 5$ are evaluated. As with H_{rec} the solution of the SCP is calculated for each available object model and the results found in Table 9 are the average of all object models.

Table 9 Results for RL-agents with changing action variants

Agent	Action	Coverage	Acquisitions	Calculation time
8	A_{2S0R}	92.20	5.26	0.0521
9	A_{3S0R}	92.16	4.63	0.0538
$B_{SCP,4}$	A_{2S0R}	96.20	**4**	506.29
$B_{SCP,5}$	A_{2S0R}	**98.17**	5	506.29
H_{rec}	A_{3C0R}	97.47	6	0.0021

Bold values indicate best achieved

Agents 8 and 9, which exhibit the best ratio of trained agents between surface coverage and the number of necessary acquisitions, are used for comparison with the benchmark algorithms. Although the trained agents perform worse compared to both analytic solution methods $B_{SCP,4}$ and $B_{SCP,5}$ in terms of surface coverage and necessary acquisitions, it can be seen that the percentage lower surface coverage is only about 4–6%. If we compare agent 9 with $B_{SCP,4}$, for example, it achieves 4.04% less surface coverage and requires 0.6 acquisitions more on average. Comparable findings can be drawn compared to the heuristic H_{rec}. In this comparison, the agents perform better. Although the agents also do not achieve the surface coverage achieved by the heuristic (5.27% less for agent 8 and 5.31% less for agent 9), they also require fewer acquisitions to do so. This can be attributed to the fact that H_{rec} only uses the rectangular bounding box to generate the views and not the object model directly. A comparison of the required calculation times reveals that the analytical solution B_{SCP} has a very high calculation time. On average, finding a view plan takes 506 s. This corresponds to about eight minutes. This is due to the simulation and subsequent evaluation of the large number of viewpoints. Such approaches are, therefore, usually used with an available object model before the actual acquisition

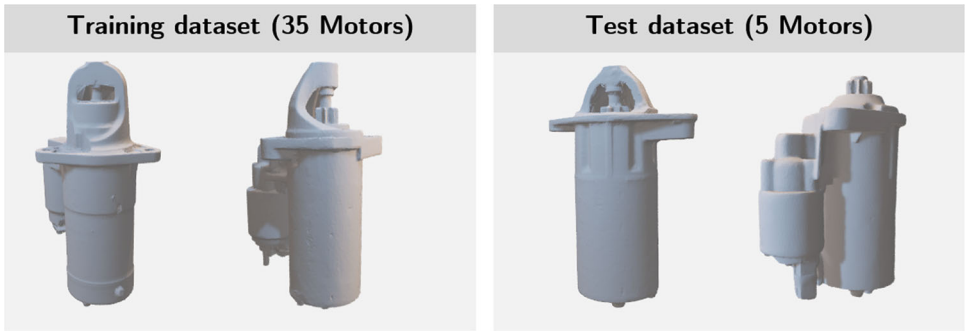

Fig. 7 Visualization of two models of each of the starter motors used in the training and test data set

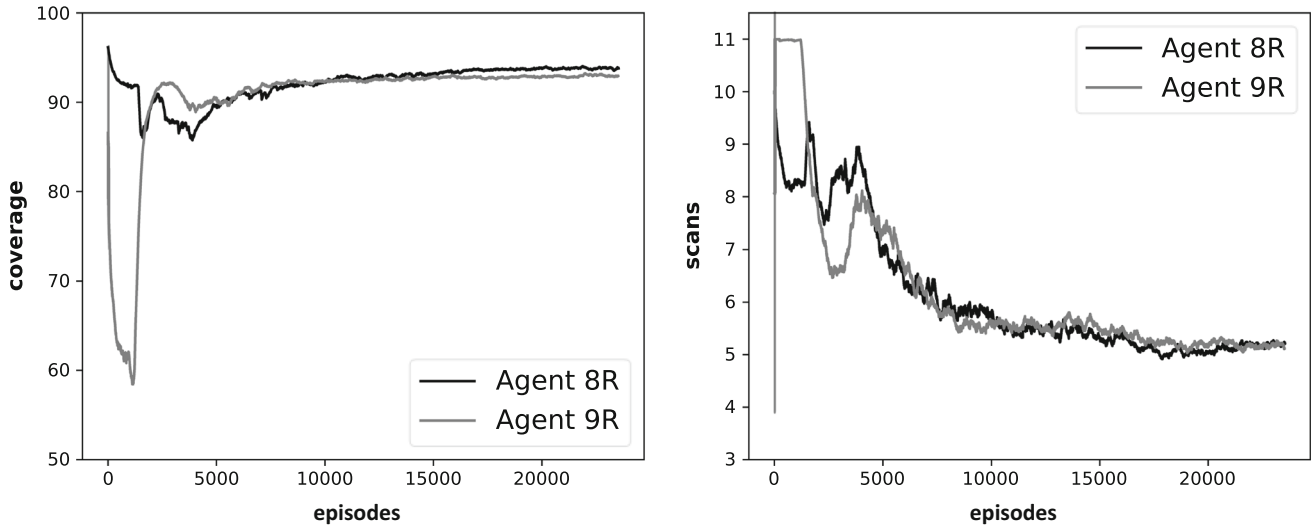

Fig. 8 Graph of the performance indicators coverage and required acquisitions (scans) for the agents 8^R and 9^R

process to generate a view plan, which then only needs to be executed for the respective inspection object at system runtime. In comparison, the trained agents 8 and 9, as well as the heuristic H_{rec}, have a low, real-time capable computing time for calculating the view plan. For the heuristic H_{rec}, only a bounding box needs to be generated for the object based on which the view plan is generated. The computing time for the view plan by the agents is calculated from the sequential action generation by the agent, simulation of an acquisition process and subsequent generation of the state, whereupon the agent selects another action until the episode ends. In doing so, accepting a 4–5 % lower surface coverage, more than one acquisition can be saved on average by using the agents, which corresponds to a reduction in the necessary execution time of a view plan. Agents 8 and 9 thus offer a good trade-off between surface coverage and the necessary acquisitions compared to H_{rec}.

Answer: *Although the trained agents perform slightly worse in comparison to the benchmark heuristic and analytic solution* in terms of overall surface coverage, it can be noted that a model-based approach using trained agents can achieve comparable results and is able to provide a good tradeoff between surface coverage and needed acquisitions for run-time capable view planning.

Training and testing using real starter motor models

Question: *Can the results be reproduced with real starter motor models that the agent has never seen?*

The results of the modeling so far have shown the applicability of the RL to the application problem presented. These results have been generated for synthetically created starter engine models. In this section, the two best-performing agents (Agent 8 and Agent 9) using synthetically created starter engine models are trained and subsequently tested using real motor models. Using a hand-held laser scanner (Zeiss T-Scan), 40 real STL models of starter motors were generated. 35 of these motor models are used to train the Agents 8^R and 9^R based on the configuration of Agent 8 and Agent 9, after which five of these models are used to test the Agents. Figure 7 shows two of the generated starter engine models of the training and test dataset.

The curves of the surface coverage and the number of acquisitions required can be found in Fig. 8. The results of agents 8^R and 9^R show agreement with the findings of

Table 10 Overview of results for training and test performance on real starter motor models

	Training performance		Test performance	
Agent	Coverage	Acquisitions	Coverage	Acquisitions
8^R	93.96	5.37	93.64	5.30
9^R	93.09	5.05	93.28	5.50
$B^R_{SCP,4}$	–	–	96.87	4
$B^R_{SCP,5}$	–	–	98.31	5
H^R_{rec}	–	–	97.03	6

Agents 8 and 9 using synthetic starter motors. Agents 8^R and 9^R exhibit similar learning behavior and perform similarly regarding the performance indicators coverage and needed acquisitions.

A comparison of the quantitative parameters (Tables 9 and 10) shows that the Agents 8^R and 9^R perform similarly to the Agents 8 and 9 with regard to the performance indicators coverage and needed acquisitions. Only the average number of necessary acquisitions has increased for Agent 9^R compared to Agent 9. It can, therefore, be concluded that the results from the training of agents with the synthetically generated starter motors can be transferred to the use case of real models of starter motors.

Comparing the performance indicators of the agents 8^R and 9^R in the training and test case, it can be stated that these performance indicators differ only slightly from those of the training case, both in the case of surface coverage and with regard to the acquisitions required for this in the test case. On average, the surface coverage in the test case is slightly smaller than in the training case. In addition, agent 9^R requires slightly more acquisitions on average in the test case than in the training case. Overall, it can be stated that the agents 8^R and 9^R provide a good solution for the VPP even on unseen real starter motors to be inspected. The results obtained with the synthetic data set can also be confirmed when comparing the heuristics and the analytical solutions. The analytical solutions $B^R_{SCP,4}$ $B^R_{SCP,5}$ show a better performance than both agents 8^R and 9^R, but they are not real-time capable. The comparison with the heuristic H^R_{rec} again provides a higher surface coverage, but the number of acquisitions required is higher.

Answer: *It can be shown that the results of agent modeling using synthetic engine models can be transferred to the case of real engine models. In addition, it has been shown that the trained agents* 8^R *and* 9^R *can also be used to solve the VPP for unseen real motor models.*

Conclusion and outlook

Specially trained workers still carry out the initial visual inspection in remanufacturing. These workers inspect the product from different angles to gain an impression of the product's condition and detect all possible defects on the entire product surface. Automating such inspection processes with the help of a robot-based inspection system can reduce the costs of inspection and, therefore, enable remanufacturing, especially in high-wage countries. In this paper, the challenges of solving the visual inspection use case on an automated system have been formalized to the visual acquisition planning problem. It has been deduced that RL is a promising approach to deal with object models missing at runtime, only getting fed the information of the object to inspect that has been acquired so far. In the methodological part of this work, a framework consisting of a simulation environment to replicate the acquisition process of a three-dimensional acquisition system and a module for variable generation of RL agents is presented. These configured RL agents can interact with the simulation environment to learn strategies to solve acquisition planning problems. The results have shown that RL can generally solve well-known vision planning problems (NBV, VPP, CPP).

Detailed analyses have shown that the SAC in combination with a feature extractor of Yuan et al. (2018)'s PCN is best suited for the problems to be solved. Successful training of the RL agents was possible both with and without knowledge of the geometrical model. This result underlines the applicability of RL in use cases of visual planning, such as remanufacturing, where no object model is available. Various alternative definitions of reward signals and their influence on agent behavior were shown. Dense reward signals were shown to outperform sparse reward signals. The best-performing agents were compared to benchmark algorithms and trained and tested on real starter engine models. Results have shown comparable results to state-of-the-art methods and the applicability of the agents to perform view planning on unseen starter engines.

Despite the positive results achieved, three key limitations persist. First, learning behavior was only possible when explicitly using prior knowledge in the form of mapping the agents' output resembling the pose of the acquisition system to a spherical surface and the automated orientation to the center of the object. Second, the agents only converged using a dense reward signal. The credit assignment problem using a sparse reward signal has, therefore, proven to be too difficult for the agent, possibly resulting in a suboptimal optimum the agents converged to. Last, the optimization of multiple objectives (in the case of this work the surface coverage maximization and the minimization of travel distances between subsequent acquisition poses) has not been proven to work.

Therefore, in future works, two major research branches will be pursued. First, an implementation of the theoretically obtained results on a real inspection system will be examined. Therefore, a robot simulation must be integrated into the existing simulation, allowing the detailed evaluation of travel distances and the reachability of the poses determined by the agents. In this way, the realism of the proposed simulation framework can be further increased. In this context, extending arbitrary object models must also be pursued. From the authors' point of view, the mapping of the agents' outputs to coordinates on a spherical surface is not sufficient for this purpose, since for arbitrary objects more freedom is needed for the positioning of the acquisition system. However, in the present work, it has been shown that the agents have insufficient learning behavior without the inclusion of the mapping of the agents' output to a spherical surface. Therefore, secondly, algorithmic improvements and sophisticated algorithms have to be investigated. This includes algorithms for multi-objective optimization to jointly optimize a range of optimization criteria such as object surface coverage and travel paths. To enable this, in contrast to the model-free RL algorithms used in this work, the usage and evaluation of model-based RL algorithms might be of interest. Further enhancements would be using imitation learning or inverse RL to learn strategies from an expert. This work's modeling results can play a key role in defining the state, action, and reward space for said research paths. Furthermore, it has to be noted that the results obtained were only achieved for one product group. In further works, the agent behavior should be investigated if several product groups and their variants are to be inspected. The theoretical findings and the simulation environment developed in this thesis provide the necessary basis for addressing these research deficits.

Author Contributions All authors assisted in the conception of the approaches presented in this work. Implementing the methodological approach by programming and validating the proposed simulation framework by Jan-Philipp Kaiser, Jonas Gäbele, Dominik Koch, and Jonas Schmid. The writing of this work was significantly driven by Jan-Philipp Kaiser, Jonas Gäbele, and Dominik Koch, supported by Florian Stamer and Gisela Lanza. Florian Stamer and Gisela Lanza provided the technical and scientific guidance of the work. Jan-Philipp Kaiser wrote the first draft of the manuscript, and all authors commented on previous versions. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was done in the project AgiProbot. AgiProbot is funded by the Carl Zeiss Foundation.

Declarations

Conflict of interest The authors have no Conflict of interest to declare that are relevant to the content of this article.

References

Achlioptas, P., Diamanti, O., Mitliagkas, I., et al. (2018). Learning representations and generative models for 3D point clouds. *Proceedings of the 35th International Conference on Machine Learning, 80*, 40–49.

Arai, T., Aiyama, Y., Maeda, Y., et al. (2000). Agile assembly system by plug and produce. *CIRP Annals, 49*(1), 1–4. https://doi.org/10.1016/S0007-8506(07)62883-2

Ashutosh, K., Kumar, S., Chaudhuri, S. (2022). 3d-nvs: A 3d supervision approach for next view selection. In *Proceedings of the 26th International Conference on Pattern Recognition (ICPR)* (pp. 3929–3936). https://doi.org/10.1109/ICPR56361.2022.9956377

Banta, J. E., Zhien, Y., Wang, X. Z., et al. (1995). Best-next-view algorithm for three-dimensional scene reconstruction using range images. *Intelligent Robots and Computer Vision XIV: Algorithms, Techniques, Active Vision, and Materials Handling, 2588*, 418–429. https://doi.org/10.1117/12.222691

Berner, C., Brockman, G., Chan, B., et al. (2019). Dota 2 with large scale deep reinforcement learning. arXiv preprint https://arxiv.org/abs/1912.06680,

Connolly, C. (1985). The determination of next best views. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (pp. 432–435). https://doi.org/10.1109/ROBOT.1985.1087372

CoremanNet. (2022). Bosch Core acceptance criteria for starter motors. Retrieved June 24, 2024, from https://www.coremannet.com/assets/docs/return-criteria/new-2019/Starter.pdf

Daniel, V., & Guide, R. (1997). Scheduling with priority dispatching rules and drum-buffer-rope in a recoverable manufacturing system. *International Journal of Production Economics, 53*(1), 101–116. https://doi.org/10.1016/S0925-5273(97)00097-2

Dawson-Haggerty, et al. (2019). trimesh. Retrieved June 24, 2024, from https://trimsh.org/

Deinzer, F., Derichs, C., Niemann, H., et al. (2009). A framework for actively selecting viewpoints in object recognition. *International Journal of Pattern Recognition and Artificial Intelligence, 23*(04), 765–799. https://doi.org/10.1142/S0218001409007351

Devrim Kaba, M., Gokhan Uzunbas, M., Nam Lim, S. (2017). A reinforcement learning approach to the view planning problem. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 6933–6941). https://doi.org/10.1109/CVPR.2017.541

DIN 31051:2019-06. (2019). Fundamentals of maintenance. Beuth Verlag GmbH, Berlin, https://doi.org/10.31030/3048531, Deutsches Institut für Normung e.V. (DIN)

DIN EN 13306:2018-02. (2018). Maintenance - Maintenance terminology; Trilingual version. Beuth Verlag GmbH, Berlin, https://doi.org/10.31030/2641990, Deutsches Institut für Normung e.V. (DIN)

Errington, M., & Childe, S. J. (2013). A business process model of inspection in remanufacturing. *Journal of Remanufacturing, 3*, 1–22. https://doi.org/10.1186/2210-4690-3-7

Haarnoja, T., Zhou, A., Abbeel, P., et al. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of Machine Learning Research (PMLR)* (Vol. 80).

Hu, S., Zhu, X., Wang, H., et al. (2008). Product variety and manufacturing complexity in assembly systems and supply chains. *CIRP Annals, 57*(1), 45–48. https://doi.org/10.1016/j.cirp.2008.03.138

Hu, S., Ko, J., Weyand, L., et al. (2011). Assembly system design and operations for product variety. *CIRP Annals, 60*(2), 715–733. https://doi.org/10.1016/j.cirp.2011.05.004

Huang, Z., Yu, Y., Xu, J., et al. (2020) Pf-net: Point fractal network for 3d point cloud completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7659–7667). https://doi.org/10.1109/cvpr42600.2020.00768

Jing, W., Goh, C. F., Rajaraman, M., et al. (2018). A computational framework for automatic online path generation of robotic inspection tasks via coverage planning and reinforcement learning. *IEEE Access, 6*, 54854–54864. https://doi.org/10.1109/ACCESS.2018.2872693

Junior, M. L., & Filho, M. G. (2012). Production planning and control for remanufacturing: Literature review and analysis. *Production Planning & Control, 23*(6), 419–435. https://doi.org/10.1080/09537287.2011.561815

Khan, A., Mineo, C., Dobie, G., et al. (2021). Vision guided robotic inspection for parts in manufacturing and remanufacturing industry. *Journal of Remanufacturing, 11*(1), 49–70. https://doi.org/10.1007/s13243-020-00091-x

Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*(11), 1238–1274. https://doi.org/10.1007/978-3-642-27645-3_18

Korbach, C., Solbach, M. D., Memmesheimer, R., et al. (2021). Next-best-view estimation based on deep reinforcement learning for active object classification. arXiv preprint https://arxiv.org/abs/2110.06766

Koren, Y. (2010). *The global manufacturing revolution: Product-process-business integration and reconfigurable systems* (Vol. 80). Wiley.

Kuhnle, A., Schäfer, L., Stricker, N., et al. (2019). Design, implementation and evaluation of reinforcement learning for an adaptive order dispatching in job shop manufacturing systems. *Procedia CIRP, 81*, 234–239. https://doi.org/10.1016/j.procir.2019.03.041

Kuhnle, A., Kaiser, J. P., Theiß, F., et al. (2021). Designing an adaptive production control system using reinforcement learning. *Journal of Intelligent Manufacturing, 32*, 855–876. https://doi.org/10.1007/s10845-020-01612-y

Kurilova-Palisaitiene, J., Sundin, E., & Poksinska, B. (2018). Remanufacturing challenges and possible lean improvements. *Journal of Cleaner Production, 172*, 3225–3236. https://doi.org/10.1016/J.JCLEPRO.2017.11.023

Landgraf, C., Meese, B., Pabst, M., et al. (2021). A reinforcement learning approach to view planning for automated inspection tasks. *Sensors (Basel, Switzerland), 21*(6), 2030. https://doi.org/10.3390/s21062030

Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al. (2015). Continuous control with deep reinforcement learning. arXiv preprint https://arxiv.org/abs/1509.02971

Mehrabi, M. G., Ulsoy, A. G., & Koren, Y. (2000). Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent manufacturing, 11*, 403–419. https://doi.org/10.1023/A:1008930403506

Mendoza, M., Vasquez-Gomez, J. I., Taud, H., et al. (2020). Supervised learning of the next-best-view for 3d object reconstruction. *Pattern Recognition Letters, 133*, 224–231. https://doi.org/10.1016/j.patrec.2020.02.024

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2013). Playing atari with deep reinforcement learning. arXiv preprint https://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533. https://doi.org/10.1038/nature14236

Monica, R., & Aleotti, J. (2021). A probabilistic next best view planner for depth cameras based on deep learning. *IEEE Robotics and Automation Letters, 6*(2), 3529–3536. https://doi.org/10.1109/LRA.2021.3064298

van Otterlo, M., & Wiering, M. (2012). Reinforcement learning and Markov decision processes. *Reinforcement Learning: State-of-the-Art*. https://doi.org/10.1007/978-3-642-27645-3_1

Pan, S., Hu, H., & Wei, H. (2022). Scvp: Learning one-shot view planning via set covering for unknown object reconstruction. *IEEE Robotics and Automation Letters, 7*(2), 1463–1470. https://doi.org/10.1109/LRA.2022.3140449

Panzer, M., & Bender, B. (2021). Deep reinforcement learning in production systems: A systematic literature review. *International Journal of Production Research*. https://doi.org/10.1080/00207543.2021.1973138

Peng, X. B., Andrychowicz, M., Zaremba, W., et al. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3803–3810). https://doi.org/10.1109/ICRA.2018.8460528

Peuzin-Jubert, M., Polette, A., Nozais, D., et al. (2021). Survey on the view planning problem for reverse engineering and automated control applications. *Computer-Aided Design, 141*, 1–22.

Potapova, S., Artemov, A., Sviridov, S., et al. (2020). Next best view planning via reinforcement learning for scanning of arbitrary 3d shapes. *Journal of Communications Technology and Electronics, 65*, 1484–1490. https://doi.org/10.1134/S1064226920120141

Qi, C. R., Su, H., Mo, K., et al. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 652–660). https://doi.org/10.1109/CVPR.2017.16

Raffin, A., Hill, A., Gleave, A., et al. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research, 22*(268), 1–8.

Ridley, S. J., Ijomah, W. L. (2015) *Pre-processing inspection—a worthwhile activity for remanufacturers*

Robotis, A., Boyaci, T., & Verter, V. (2012). Investing in reusability of products of uncertain remanufacturing cost: The role of inspection capabilities. *International Journal of Production Economics, 140*(1), 385–395. https://doi.org/10.1016/j.ijpe.2012.04.017

Schlüter, M., Lickert, H., Schweitzer, K., et al. (2021). Ai-enhanced identification, inspection and sorting for reverse logistics in remanufacturing. *Procedia CIRP, 98*, 300–305. https://doi.org/10.1016/j.procir.2021.01.107

Scholz-Reiter, B., & Freitag, M. (2007). Autonomous processes in assembly systems. *CIRP Annals, 56*(2), 712–729. https://doi.org/10.1016/j.cirp.2007.10.002

Schulman, J., Levine, S., Abbeel, P., et al. (2015). Trust region policy optimization. arXiv preprint arXiv: org/abs/1502.05477

Schötz, S., Butzer, S., Molenda, P., et al. (2017). An approach towards an adaptive quality assurance. *Procedia CIRP, 63*, 189–194. https://doi.org/10.1016/j.procir.2017.03.096

Scott, W., Roth, G., & Rivest, J. F. (2003). View planning for automated three-dimensional object reconstruction and inspection. *ACM Computer Survey, 35*, 64–96. https://doi.org/10.1145/641865.641868

Scott, W. R. (2009). Model-based view planning. *Machine Vision and Applications, 20*(1), 47–69. https://doi.org/10.1007/s00138-007-0110-2

Shen, L., Tao, H., Ni, Y., et al. (2023). Improved yolov3 model with feature map cropping for multi-scale road object detection. *Measurement Science and Technology, 34*(4), 045406. https://doi.org/10.1088/1361-6501/acb075

Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017). Mastering the game of go without human knowledge. *Nature, 550*(7676), 354–359. https://doi.org/10.1038/nature24270

Sundin, E. (2004). *Product and process design for successful remanufacturing*. Dissertation, Linköping University.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tolio, T., Bernard, A., Colledani, M., et al. (2017). Design, management and control of demanufacturing and remanufacturing systems. *CIRP Annals, 66*(2), 585–609. https://doi.org/10.1016/j.cirp.2017.05.001

Vasquez-Gomez, J. I., Troncoso, D., Becerra, I., et al. (2021). Next-best-view regression using a 3d convolutional neural network. *Machine Vision and Applications, 32*, 1–14. https://doi.org/10.1007/s00138-020-01166-2

Vongbunyong, S., Chen, W. H., Vongbunyong, S., et al. (2015). *Disassembly automation*. Springer. https://doi.org/10.1007/978-3-319-15183-0

Wang, X., Ang, M. H., Lee, G. H. (2020). Cascaded refinement network for point cloud completion. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 787–796). https://doi.org/10.1109/cvpr42600.2020.00087

Wen, X., Li, T., Han,, Z., et al. (2020). Point cloud completion by skip-attention network with hierarchical folding. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1936–1945). https://doi.org/10.1109/cvpr42600.2020.00201

Wu, C., Zhou, K., Kaiser, J. P., et al. (2022). Motorfactory: A blender add-on for large dataset generation of small electric motors. *Procedia CIRP, 106*, 138–143. https://doi.org/10.1016/j.procir.2022.02.168

Yuan, W., Khot, T., Held, D., et al. (2018). Pcn: Point completion network. In *Proceedings of the 2018 International Conference on 3D Vision, Processing, Visualization and Transmission (3DIMPVT)* (pp. 728–737). https://doi.org/10.1109/3DV.2018.00088

Zeng, R., Wen, Y., Zhao, W., et al. (2020). View planning in robot active vision: A survey of systems, algorithms, and applications. *Computational Visual Media, 6*, 225–245. https://doi.org/10.1007/s41095-020-0179-3