OpenDR —
Open Deep Learning Toolkit for Robotics

Project Start Date: 01.01.2020
Duration: 48 months
Lead contractor: Aristotle University of Thessaloniki

Deliverable D5.2: Second report on deep robot action and decision making

Date of delivery: 31 December 2021

Contributing Partners: TUD, ALU-FR, AU, TAU
Version: v1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871449.

Title	**D5.2: Second report on deep robot action and decision making**
Project	**OpenDR** (ICT-10-2019-2020 RIA)
Nature	Report
Dissemination Level:	**PU**
Authors	Bas van der Heijden (TUD), Jelle Luijkx (TUD), Laura Ferranti (TUD), Jens Kober (TUD), Robert Babuška (TUD), Halil Ibrahim Ugurlu (AU), Erdal Kayacan (AU), Lukas Hedegaard Morsing (AU), Amir Mehman Sefat (TAU), Roel Pieters (TAU), Daniel Honerkamp (ALU-FR), Tim Welschehold (ALU-FR), Abhinav Valada (ALU-FR), Wolfram Burgard (ALU-FR), Anastasios Tefas (AUTH), Nikos Nikolaidis (AUTH)
Lead Beneficiary	TUD (Technische Universiteit Delft)
WP	5
Doc ID:	OPENDR_D5.2.pdf

Document History

Version	**Date**	**Reason of change**
v0.1	13/10/2021	Deliverable structure template ready
v0.2	13/10/2021	Deliverable ready for internal review
v1.0	15/12/2021	Deliverable ready for submission

Contents

Executive Summary

This document presents the status of the work performed for **WP5–Deep robot action and decision making**. WP5 consists of four main tasks, that are *Task 5.1–Deep Planning*, *Task 5.2–Deep Navigation*, *Task 5.3–Deep Action and Control*, and *Task 5.4–Human Robot Interaction*.

After a general introduction that provides an overview of the individual chapters with a link to the main objectives of the project, the document dedicates a chapter to each tasks. Each chapter *(i)* provides an overview on the state of the art for the individual topics and existing toolboxes, *(ii)* details the partners' current work in each task with initial performance results, and *(iii)* describes the next steps for the individual tasks. Finally, a conclusion chapter provides a final overview of the work and the planned future work for each individual task.

1 Introduction

This document describes the work done during the second year of the project in the four major research areas of WP5 namely deep planning, deep navigation, deep action and control, and human-robot interactions.

The next sections (Sections 1.1-1.5) provide a summary of the work done so far on these three main topics and the link with the project objectives. The rest of the document is structured as follows. Chapter 2 details our work on deep planning. Chapter 3 describes our work on deep navigation. Chapter 4 presents our work on deep action and control. Chapter 5 presents our work on human robot interaction. Finally, Chapter 6 concludes this deliverable.

More details related to the implementations of the proposed methods in the OpenDR Toolkit can be found in D7.2 (WP7).

1.1 Deep Planning (T5.1)

1.1.1 Objectives

Conventional robot motion planning is based on solving individual sub-problems such as perception, planning, and control. On the other hand, end-to-end motion planning methods intend to solve the problem in one shot with less computation cost. Deep learning enables us to learn such end-to-end policies, particularly integrated with Reinforcement learning. AU introduces end-to-end motion planning methods for UAV navigation trained with Deep reinforcement learning.

1.1.2 Innovations and achieved results

AU proposed a novel end-to-end path planning algorithm (AgroRL) for multiple aerial-ground robots team for green transition in agriculture. In the proposed solution, while main operations in the field are handled by the ground vehicle, the aerial robot is responsible for re-planning a collision-free trajectory for the ground robot when the robot faces an obstacle. Deep reinforcement learning is used for training the end-to-end policy for local re-planning of the aerial robot. The agent, informed by the global trajectory, generates local plans based on depth images. Variational autoencoders are also investigated for dimension reduction of the depth images in obstacle avoidance context to speed up deep reinforcement learning and alleviate the computational complexity of the policy network. The agriculture environment is developed in the Webots open-source robot simulator. Finally, the efficiency and efficacy of the ground-aerial robot team are evaluated over a number of cluttered field scenarios. The extensive simulation and real-world experiments demonstrate that the use of an aerial robot enhances the ground robot's capabilities significantly compared to having further sensors on the ground robot.

1.1.3 Ongoing and future work

There are two possible future directions to improve the presented end-to-end planner. Firstly, AU aims to make the planner more lightweight to execute in real-time by providing a lighter representation learning methodology to the framework. Secondly, AU plans to improve multi-robot planning infrastructure by introducing UGV's actions to the learning framework.

1.2 Deep Navigation (T5.2)

1.2.1 Objectives

Learning based approaches have shown to be well suited to solve navigation tasks across diverse environments and platforms, including autonomous vehicles, video games and robotics. Particularly deep learning and reinforcement learning approaches have shown to work well with the complex, high-dimensional inputs of real-world environments. Navigation tasks involve both long-horizon goals that require long-term planning as well as local, short-term decision making such as traversing unknown terrain or avoiding static and dynamic obstacles. As a result both the decomposition of the problem into different components and levels of abstraction as well as the combination of traditional optimization and planning approaches with learned modules are very promising approaches.

1.2.2 Innovations and achieved results

ALU-FR has developed a novel mobile navigation and manipulation approach for wheeled robots. While existing approaches most commonly separate such tasks into point-goal navigation and static manipulation, this is both inefficient and restrictive for certain tasks such as door opening. We decompose mobile manipulation into an arbitrary end-effector motion and a reinforcement learning agent controlling the base. This enables to very easily specify tasks such as pick & place or door opening tasks as simple Cartesian motions. At the same time this formulation leaves the trained agent agnostic to the exact task, allowing to generalise to unseen tasks in a zero-shot manner. The approach is demonstrated to achieve high success rates on a real PR2 robot and the tool is integrated into the OpenDR repository. ALU-FR has proposed a novel dynamic audio-visual navigation benchmark as well as an approach which strongly improves the generalization to unheard sounds on static audio-visual navigation.

1.2.3 Ongoing and future work

We are currently extending our mobile manipulation approach to incorporate obstacle avoidance, enabling it to easily complete tasks within cluttered, human-centered environments. Furthermore, we are aiming to demonstrate it's effectiveness in navigating dynamic obstacles, with promising initial results.

1.3 Deep Action and Control (T5.3)

1.3.1 Objectives

Model-based reinforcement learning (RL) is well-suited for robotics due to its sample complexity. In the high-dimensional case, the policy must be learned in a lower-dimensional latent space to fully exploit this data efficiency. For the learned policy to run on a robotic platform with limited computational resources, the latent dynamics model must be compatible with fast control algorithms. Therefore, TUD aims to find a lightweight and sample efficient control method that can deal with high-dimensional observations.

Modern quadrotors are agile and can perform complex tasks in difficult-to-reach places. Quadrotor flight and maneuvers are commonly controlled by proportional integral derivative (PID) control or model predictive control (MPC). Although these methods are adequate for set-point or trajectory tracking, they fall short when it comes to more complicated maneuvers that exceed the linearization range or require long prediction horizons. One such maneuver is the landing on an inclined surface, which is relevant for applications like delivery, maintenance, or surveillance. To aid in safe operation with quadcopters, TUD aims to find a method for safely learning an inclined landing policy.

OpenAI Gym [8] is the standard for training and evaluation of reinforcement learning algorithms. However, developing Gym environments for robot control tasks is an inefficient process that requires expertise, since it is all but trivial to synchronise actions and observations while having asynchronous input and output streams for sensors and actuators that run at different frequencies. In order to facilitate this process TUD aims to develop EAGERx, a toolkit that bridges the gap between OpenAI gym and robot, both real and simulated. The EAGERx toolkit aims to separate everything that is engine-specific from everything that is engine-agnostic, such that that environments can be reused for different simulators and even when switching from simulated to real robots.

Despite the recent successes of deep neural networks in robotics, it has not yet become a standard component in control design toolbox. This is due to several limitations imposed by the practice/requirement of manually tuning a large number of hyperparameters. Optimal tuning of these parameters require significant expertise. To simplify the tuning process and to save the rare resource of experts, TUD aims to develop a holistic hyperparameter tuning tool, which is compatible with all learner classes from the OpenDR toolkit. This tool provides integration of the OpenDR toolkit with the existing hyperparameter tuning framework Optuna [3].

Learning-based grasping models typically require a vast amount of training data and training time to train an effective grasp pose detector. Alternatively, small non-generic grasp models have been proposed that are tailored to specific objects by, for example, directly predicting the object's location in 2/3D space, and determining suitable grasp poses by post processing. In both cases, data generation is a bottleneck, as it has to be separately collected for each individual object. Moreover, most works consider objects in household scenarios which makes them unsuitable for industrial settings as the physical properties of the objects are different. In this work, TAU aims to tackle these issues and propose a light-weight grasping pipeline that is divided in four main steps: 1. single object demonstration, 2. object data augmentation, 3. grasp model training and 4. object grasping action.

1.3.2 Innovations and achieved results

TUD focused on the use of Koopman theory combined with deep learning for efficient control of robotic systems. A novel model based agent, DeepKoCo, was presented, that learns a latent Koopman representation from images. This representation allows DeepKoCo to plan efficiently using linear control methods, such as linear model predictive.

TUD presented a method for learning a quadcopter inclined landing policy in simulation that directly transferred to the real world. As the policy was learned entirely in simulation, it was not only efficiently learned but also completely safe as exploration was only performed in simulation.

TUD develops the EAGERx toolkit that bridges the gap between OpenAI gym and robotics. It facilitates the process of creating Gym environments for robot control tasks and will provide synchronisation of actions and observations even when sensors and actuators are running at different rates and are providing asynchronous data streams. Also, by isolating engine-specific definitions, EAGERx allows to reuse environments for different simulators and even when switching from a simulated to a real robot.

TUD develops a hyperparameter tuning tool that provides integration of the Optuna hyperparameter tuning framework within the OpenDR toolkit. The tool was designed with the aim to be compatible with all learner classes of the OpenDR toolkit.

TAU proposed an approach that utilizes state of the art keypoint detector ”Keypoint-RCNN” to develop a grasping pipeline that is able to learn the grasp pose demonstration by the user. In this work, our grasping pipeline generates an augmented training dataset from a single user demonstration which consist of a few images of the object from different camera views. These images then are heavily augmented and utilized to train a keypoint detector and the predicted keypoints are then post-processed and translated to 3D grasp poses. Four different vision-based methods are evaluated for deriving the relative rotation of the object with respect to the reference/target frame alongside an object detection module. Evaluation considers the grasping of different industrial and 3D printed objects with an industrial collaborative manipulator, and shows >90% success rate.

1.3.3 Ongoing and future work

Concerning the use of Koopman theory combined with deep learning for control of robotic systems, as part of future work, TUD plans to investigate if DeepKoCo can be extended to manipulators. If convincing results are achieved, TUD intents to include the extended method into the toolkit. Else, TUD will focus on more promising research directions.

The presented method for learning a quadcopter inclined landing policy by the TUD, provides valuable insights on sim2real transfer of quadcopter policies that could prove valuable for the quadcopters used in the agricultural use-case. This will be further investigated in future work.

The development of the EAGERx toolkit will be continued and additional features will be added in order to improve the usability, such as a graphical user interface, validity checks and extensive documentation. Also, the toolkit will be further evaluated and more experiments will follow. Furthermore, support for more hardware will be provide.

The hyperparameter tuning tool will be evaluated on a practical problem in order to validate its usefulness. Also, hyperparameter and search space definitions will be provided for existing tools in order to facilitate the hyperparameter tuning process for users of the OpenDR toolkit.

The proposed single demonstration grasping method by TAU demonstrates ability to perform in agile production use cases. Using 3D mesh and depth information and infusing more sensory data would increase the robustness and mitigates the limitations of 2D RGB solutions. On the other hand, the quality of grasp is also of high importance in agile production use-case. TAU will study this possibilities in the future.

1.4 Human Robot Interaction (T5.4)

1.4.1 Objectives

The interaction between human and robot can serve several functions. When considering a collaborative scenario, the robot can assist a person by taking heavy, dirty or repetitive tasks, relieving the person to a more supervisory or coordinating role. In addition, robots can learn to interact with humans but also from interactions with humans. In both cases understanding human intentions and behavior is crucial, which can by enabled by motion capturing and other ways of instrumenting the human partner. Typically, the human state and intentions are very ambiguous and uncertain (e.g., forces that are crucial for successful completion cannot be estimated from video) and simulations of the human partner and its behavior are unlikely to be very realistic, compounding the uncertainties. Fusing information from multiple modalities will allow the robot to disambiguate. For DRL of human-robot interaction tasks require approaches that can deal with large uncertainties by making optimal use of fused, multimodal perception. Learning from demonstrations, for example, often relies on kinesthetic teach-in, videos, VR, or textual instructions. The objectives of this task are therefore to utilize the OpenDR toolkit towards human-robot interaction and enable efficient and effective collaboration.

1.4.2 Innovations and achieved results

TAU has studied human-robot interaction in the context of collaborative manufacturing, where a small collaborative robot assists an operator in assembly tasks. In these scenarios, robot programming has mostly focused on automated robot motions and interactive tasks or coordination between human and robot still requires additional developments. For example, the selection of which tasks or actions a robot should do next might not be known beforehand or might change at the last moment. Within a human-robot collaborative setting, the coordination of complex shared tasks, is therefore more suited to a human, where a robot would act upon requested commands. In this work we explored the utilization of commands to coordinate a shared task between a human and a robot, in a shared work space. Based on a known set of higher-level actions (e.g., pick-and-placement, hand-over, kitting) and the commands that trigger them, both a speech-based and graphical command-based interface are developed to investigate its use. While speech-based interaction might be more intuitive for coordination, in industrial settings background sounds and noise might hinder its capabilities. The graphical command-based interface circumvents this, while still demonstrating the capabilities of coordination. The developed architecture follows a knowledge-based approach, where the actions available to the robot are checked at runtime whether they suit the task and the current state of the world. Experimental results on industrially relevant assembly, kitting and hand-over tasks in a laboratory setting demonstrate that graphical command-based and speech-based coordination with high-level commands is effective for collaboration between a human and a robot. These results are evaluated with metrics as typical in manufacturing, such as human and robot idle time (H-IDL and R-IDL), concurrent activity (C-ACT) and functional delay (F-DEL), which demonstrate benefits towards the coordinated actions, as compared to traditional approaches.

1.4.3 Ongoing and future work

As ongoing work, the human-robot collaboration scenario has utilized the tools developed in WP3 (speech recognition). Initial results look promising and will be extended and evaluated as future work. This includes the recognition of words relevant to the collaborative scenario (i.e., agile production) and the recognition of word pairs (i.e., action-target). In addition, multi-modal perception tools will be included to enable more precise commanding by, for example, combing speech and gesture/object recognition.

1.5 Connection to Project Objectives

The work performed within WP5, as summarized in the previous subsections, perfectly aligns with the project objectives. More specifically, the conducted work progressed the state-of-the-art towards meeting following objectives of the project:

O2.c *To provide lightweight deep learning methods for deep robot action and decision making, namely:*

- O2.c.i Deep reinforcement learning (RL) and related control methods.
  - ∗ TUD contributed to this objective as detailed in parts of Chapter 4. A lightweight and sample efficient method based on Koopman theory was presented. This approach allows one to efficiently control robotic systems using high-dimensional observations that are possibly contaminated with task-irrelevant dynamics. Secondly, TUD presented an approach to learn a quadcopter inclined landing policy in simulation that directly transferred to the real-world.
  - ∗ Finding suitable hyperparameters can be a tiresome — but also essential — part of developing deep reinforcement learning methods. TUD has developed hyperparameter tuning functionality for all deep learning tools in the OpenDR toolkit that allows users to easily find the right hyperparemeters.
- O2.c.ii Deep planning and navigation methods that can be trained in end-to-end fashion.
  - ∗ ALU-FR developed a mobile navigation tool for wheeled robots. Given an arbitrary motion for the end-effector, the tool uses reinforcement learning to produce corresponding base motions that make these motions feasible. The tool is lightweight and can run on standard CPUs. The module has been tested on a real PR2 and is implemented for both differential drive robots (PAL TiaGo) and omnidirectional robots (PR2, Toyota HSR). ALU-FR furthermore developed an approach for audio-visual navigation that achieves state-of-the-art results in the generalisation to unheard sounds.
  - ∗ AU contributed to this objective as described in Chapter 2. An end-to-end planner for a quadrotor UAV is provided for collision avoidance in collaboration with multiple ground vehicles. The DRL agent, informed by the global trajectory, generates actions as local position plans based on depth images from the UAV. The efficiency and efficacy of the ground-aerial robot team are evaluated over several cluttered field scenarios.
- O2.c.iii Enable robots to decide on actions based on observations in WP3, as well as to learn from observations
  - ∗ TUD has contributed to this objective with the pre-processing functionality in the EAGERx toolkit. This functionality allows the user to use the perception algorithms in WP3 as a pre-processing step.
  - ∗ TAU has contributed to this objective as described in Chapter 4. A novel single demonstration grasping (SDG) model has been developed, refined and evaluated that takes as input a single image of an object and generates the required training data to train a grasping model. Several grasp detection methods are evaluated for grasping object relevant in the agile production use case. In addition, TAU has contributed to this objective as described in Chapter 5, where speech recognition tools from WP3 are utilized to command robot actions in a human-robot collaborative scenario.
- O2.c.iv Enable efficient and effective human robot interaction
  - ∗ TAU has contributed to this objective as described in Chapter 5. A collaborative scenario between human and robot was developed, inspired from the Agile Pro-

duction use case, in which the robot provides assistance to the human by kitting tasks and object hand-over tasks. Commands from human to robot are utilized to coordinate the collaboration, by both a graphical interface and speech recognition from WP3. Results are evaluated by metrics that are typical for industrial manufacturing (human and robot idle time (H-IDL and R-IDL), concurrent activity (C-ACT) and functional delay (F-DEL)), which demonstrate efficient and effective human-robot interaction.

2 Deep Planning

2.1 End-to-end Path Planning of Air-Ground Multi-Robot Team

2.1.1 Introduction and objectives

Robot teams are commonly used for their ability to work on a single task collaboratively. They are especially advantageous when the robots have different capabilities and/or operate in different domains. For instance, a collaboration between an aerial and a ground robot might be helpful: long-term and close contact tasks are assigned to the ground robot due to its relatively higher power source, and observation tasks are assigned to the aerial vehicle due to its better field of view.

In this study, a collaborative solution is proposed in an agricultural field to leverage the efficiency of the agricultural operation with artificial intelligence (AI)-based navigation methods. Particularly, unmanned ground vehicles (UGVs) are responsible for field operations, *e.g.*, seeding, and an unmanned aerial vehicle (UAV) deals with the navigation performance enhancement of the team. Since machinery and equipment are the major costs in agricultural operations and ground vehicles (*e.g.*, a tractor costs around 200K USD) are significantly more expensive than the aerial robots (*e.g.*, a typical mid-size UAV costs 10K USD), the motivation behind this study is to reduce the overall cost of the operation by also increasing the accuracy and productivity. In order to minimize the cost of the ground vehicle, the UGV is considered blind in its contribution to the planning task and accomplishes its field tasks with the given motion commands. This work focuses on the motion planning of the robot team using the information of a global path to follow and the depth camera on the UAV. In case of an obstacle occurrence on the global path, the UAV follows a collision-free path and informs the UGV about the required maneuver. In this way, the required sensor costs on the UGV is minimized.

State estimation, perception, planning, and control are conventionally considered as separate problems to be solved in autonomous robot navigation. On the other hand, recent developments in machine learning, particularly in deep reinforcement learning (DRL), enable an agent to learn various navigation tasks with only a single neural network policy. These methods are promising to solve navigation problems faster since they do not deal with the unnecessary optimization of particular problems; however, they are also hard to debug, making them hard to apply in real scenarios. In this work, as an end-to-end planning method, a policy network for collaborative agricultural planning is trained using DRL, which provides position steps using a multi-modal input, a depth camera, and global trajectory information.

While DRL methods allow solving sequential decision-making tasks by only defining a reward function, they are considered sample inefficient. They need enormous amount of training samples in order to reach a satisfactory performance. The sample inefficiency rises with the dimension of state or action spaces in the problem definition. An option to reduce the dimensionality is to encode -i.e., using a variational autoencoder (VAE)- high-dimensional data, such as the depth image. Therefore, the effect of encoding the depth image is investigated to reduce the load on the DRL algorithm and to obtain a faster policy in this study.

The contributions of this study are fourfold:

- An open-source RL framework (AgroRL) is proposed for training an end-to-end planner for a quadrotor UAV.
- VAE-based state representation is investigated for end-to-end reactive planning of UAVs.
- Multiple UGVs working in an agricultural field is integrated with the UAV agent to generate collision-free local motion plans.
- The method is evaluated with extensive experiments in a Webots-based simulation environment and demonstrated in a real-world indoor scenario.

2.1.2 Description of work performed so far

The details of this work are found in the corresponding publication that is listed below, and can be found in Appendix D:

- H. I. Ugurlu, D. Bardakci, H. X. Pham and E. Kayacan "*AgroRL: End-to-end Path Planning of Air-Ground Multi-Robot Team for Green Digital Farming*" (Submitted to ICRA 2022)

Increasing the operational efficiency of agricultural machines is essential by the use of artificial intelligence (AI)-based navigation, planning, and control algorithms to handle the increasing demand for food production without compromising sustainability. In this study, a novel end-to-end path planning algorithm (AgroRL) is proposed for multiple aerial-ground robots team for green transition in agriculture. In the proposed solution, while main operations in the field are handled by the ground vehicle, the aerial robot is responsible for re-planning a collision-free trajectory for the ground robot when the robot faces an obstacle. Deep reinforcement learning is used for training the end-to-end policy for local re-planning of the aerial robot. The agent, informed by the global trajectory, generates local plans based on depth images. Variational autoencoders are also investigated for dimension reduction of the depth images in obstacle avoidance context to speed up deep reinforcement learning and alleviate the computational complexity of the policy network. The agriculture environment is developed in the Webots open-source robot simulator. Finally, the efficiency and efficacy of the ground-aerial robot team are evaluated over a number of cluttered field scenarios. The extensive simulation and real-world experiments demonstrate that the use of an aerial robot enhances the ground robot's capabilities significantly compared to having further sensors on the ground robot.

2.1.3 Future work

One possible future research direction in this study is to implement a better representation learning methodology instead of decoupling representation learning and DRL. When the algorithm can run continuously in real-time, there is a possibility to provide lower-level control commands to the UAV, which yields a faster flight. Another future direction is to integrate multiple robots

(UAV and UGVs) in the same learning problem. Since the end-to-end planner is separated from the UGV planner, only common obstacles can be considered in the problem. However, one advantage of having a UAV is its better FOV. Combining multi-robot planning in the same problem will provide the ability to avoid obstacles in the ground without bothering the UAV.

3 Deep Navigation

3.1 Learning Kinematic Feasibility for Mobile Manipulation Through Deep Reinforcement Learning

3.1.1 Introduction and objectives

In recent years, approaches to improving the capabilities of robotic platforms to perform flexible and complex tasks in both industrial and dynamic domestic environment achieved impressive results [5, 7, 36, 34]. So far, however, most of these approaches separate navigation and manipulation due to the difficulties in planning the joint movement of the robot base and its end-effector (EE). Typically the tasks that a robot is expected to perform are linked to conditions in the task space, such as poses at which handled objects can be grasped, orientation that objects should maintain or entire trajectories that must be followed. While there are approaches to position a manipulator to fulfill various task constraints with respect to the robot's kinematics based on inverse reachability maps (IRM) [29], performing such tasks while moving the base remains an unsolved problem.

Classical planning approaches circumvent kinematic issues implicitly by exploring paths in the robot's configuration space [9]. However, this creates a number of new difficulties. First, the constraints must be transferred from the task space to the robot specific configuration space, requiring expert knowledge on the task, the robot and the environment. Furthermore, the execution of pre-planned configuration space movements in dynamic environments is not easy as minor errors in the execution of poses in the configuration space can lead to large deviations in the task space and, due to changes in the dynamic scene, adjustments to the movement might be necessary, requiring a complete re-planning in the configuration space.

3.1.2 Description of work performed so far

Details of this work can be found in the publication listed below, which is also provided in Appendix E:

- [20] D. Honerkamp, T. Welschehold and A. Valada, "*Learning Kinematic Feasibility for Mobile Manipulation Through Deep Reinforcement Learning*", in IEEE Robotics and Automation Letters, vol. 6, no. 4, pp. 6289-6296, Oct. 2021, *DOI: 10.1109/LRA.2021.3092685*.

We propose a novel approach to mobile manipulation. Given an arbitrary end-effector motion, we train a reinforcement learning agent to control the base of the robot to ensure that these motions remain kinematically feasible. This has several benefits: (i) it provides a simple way to define mobile manipulation tasks and to incorporate task constraints (ii) it provides a simple dense reward signal to learn these long-horizon tasks (iii) as the reinforcement agent is conditioned on motions of the end-effector and agnostic to the exact task, it can easily generalize to novel tasks in a zero-shot manner. The effectiveness of this approach is extensively evaluated in simulation as well as on a real PR2. It directly transfers into the real world where it effectively solves unseen tasks based on novel end-effector motions. The model is lightweight and can run on standard CPUs.

3.1.3 Future work

We are currently extending this approach to incorporate obstacle avoidance. This will enable it to easily complete tasks within cluttered, human-centered environments. For this we are incorporating a local occupancy map into the observation space. This choice ensures good generalization to unseen obstacles and environments as well as high flexibility with regard to the sensors from which the occupancy maps are constructed. We also enable the agent to control the velocity of the end-effector motions, increasing the agent's degrees of freedom and enabling it to navigate around large obstacles. Lastly, we are aiming to demonstrate it's effectiveness in navigating dynamic obstacles. In contrast to existing planning based methods, our control approach is completely dynamic and does not require a complete re-planning if the environment changes.

3.2 Audio-Visual Navigation in Complex Unmapped Environments with Moving Sounds

3.2.1 Introduction and objectives

Humans are able to very efficiently combine their senses of hearing and sight in order to navigate unknown environments. While navigation in such environments has been an important focus of embodied AI [17, 11, 40], existing work on navigation overwhelmingly relies on sensors such as vision and LiDAR, leaving out other core senses used by humans. Sound is a particularly unique modality as it reveals information beyond the visible walls and obstacles. In particular, it has been shown to provide blind people spatial navigation capability comparable to sighted people [16]. Navigation-centered approaches have shown that agents can successfully extract information from the audio signals. However, they have mostly focused on clean and distractor-free audio settings in which the only change to the audio signal comes from changes in the agent's position. Furthermore, they have struggled to generalize to unheard sounds [12, 13].

3.2.2 Description of work performed so far

We take the next steps towards more challenging scenarios. First, we introduce a novel dynamic audio-visual navigation benchmark with a moving sound source. This captures common scenarios such as a robot navigating to a person issuing commands or following pets or people in the house. We argue that this strongly increases the complexity of the task through two channels: on one hand, previous observations no longer capture the current state of the environment and the agent has to learn to update its memory accordingly. On the other hand, optimal behavior now requires not just to follow the sound intensity but proactively reason about the movement of the target to catch it efficiently. Secondly, we increase the difficulty of both static and moving sound tasks by designing complex audio scenarios with augmented, noisy and distracting sounds and show the benefits of training on these scenarios for generalization to unheard sounds. Lastly, we introduce an architecture that explicitly enables the agent to spatially fuse the geometric information inherent in obstacle maps and audio signals. We show that this leads to significant gains in the generalization to unheard sounds in both clean and complex audio scenarios.

We demonstrate these results in the SoundSpaces [12] extension to the Habitat simulator [33], which allows us to generate realistic binaural sound signals for the realistic 3D environments of the Replica [35] and Matterport3D [10] datasets. In combination, these contributions achieve improvements over the previous state-of-the-art by 53% and 29% in the success rate as well as an improvement in SPL by 58% and 39% on Replica and Matterport3D respectively, on the unheard and static AudioGoal benchmark tasks [12]. Part of this work has achieved the first place of the SoundSpaces Challenge at the CVPR 2021 Embodied AI Workshop[1]. Furthermore, it is currently under review as a conference submission.

4 Deep action and control

4.1 DeepKoCo: Efficient latent planning with a task-relevant Koopman representation

4.1.1 Introduction and objectives

Model-based reinforcement learning is well-suited for robotics due to its sample complexity. In the case of high-dimensional observations, the policy must be learned in a lower-dimensional latent space to fully exploit this data efficiency. For the learned policy to run on a robotic platform with limited computational resources, the latent dynamics model must be compatible with fast control algorithms. For this, Koopman operator theory combined with deep learning provides a promising research direction. Although a latent Koopman representation facilitates more efficient control, it still focuses the majority of the model capacity on potentially task-irrelevant dynamics that are contained in the observations. Therefore, it is of great importance to mitigate the effect of task-irrelevant dynamics in the latent representation learning. Therefore, our objective is to find a lightweight and sample efficient method based on Koopman theory to control robotic systems that lack a simple state description using high-dimensional observations that are possibly contaminated with task-irrelevant dynamics. Specifically, we aim for a control algorithm that is:

1. **Sample efficient** The learning algorithm must be sample-efficient, because real environment interactions are both time-consuming and costly in robotic systems due to wear of the equipment.

2. **Lightweight** Combine deep learning methods with Koopman theory to enable the use of lightweight linear optimal and robust control techniques in the latent space.

3. **High-dimensional observations** Enable robots to efficiently decide on actions based on high-dimensional observations (e.g. images by first mapping them to a low-dimensional space).

4. **Invariance to task-irrelevant dynamics** Learn a low-dimensional mapping that is invariant to task-irrelevant dynamics.

[1]`https://soundspaces.org/challenge`.

4.1.2 Description of work performed so far

The details of this work are found in the corresponding publication that is listed below, and can be found in Appendix A:

- [37] B. van der Heijden, L. Ferranti, J. Kober and R. Babuska "*DeepKoCo: Efficient latent planning with a task-relevant Koopman representation*", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.

This paper presents DeepKoCo, a novel model based agent that learns a latent Koopman representation from images. This representation allows DeepKoCo to plan efficiently using linear control methods, such as linear model predictive control. Compared to traditional agents, DeepKoCo learns task relevant dynamics, thanks to the use of a tailored lossy autoencoder network that allows DeepKoCo to learn latent dynamics that reconstruct and predict only observed costs, rather than all observed dynamics. As our results show, DeepKoCo achieves a similar final performance as traditional model-free methods on complex control tasks, while being considerably more robust to distractor dynamics, making the proposed agent more amenable for real-life applications.

4.1.3 Future work

We believe the presented method has merit for OpenDR, if and only if the performance we obtained with images in the pendulum task transfers to a manipulator. As of now, we have not been able to achieve a similar performance in the Reacher task [37] with images. Hence, modifications to the presented method are required, in order to be suitable for inclusion in the OpenDR toolkit. We intent to explore how to extend the current method, and include the extended method into the toolkit if we are able to show convincing results with images. Else, we will pursue more promising research direction.

4.2 Inclined Quadrotor Landing using Deep Reinforcement Learning

4.2.1 Introduction and objectives

Modern quadrotors are agile and can perform complex tasks in difficult-to-reach places. Quadrotor flight and maneuvers are commonly controlled by proportional integral derivative (PID) control or model predictive control (MPC). Although these methods are adequate for set-point or trajectory tracking, they fall short when it comes to more complicated maneuvers that exceed the linearization range or require long prediction horizons. One such maneuver is the landing on an inclined surface, which is relevant for applications like delivery, maintenance, or surveillance. To facilitate a safe inclined landing, the final attitude of the quadrotor must match the slope of the landing platform. The final state of the landing trajectory is not an equilibrium, which presents a challenge for the control design. Owing to the under-actuated nature of the system, the landing trajectory can be long and complex, with an initial motion away from the landing location. This complicates the use of standard control methods like MPC with a fixed prediction horizon and quadratic cost function. In this work, we developed a DRL approach to solve the inclined landing problem.

4.2.2 Description of work performed so far

The details of this work are found in the corresponding publication that is listed below, and can be found in Appendix B:

- [24] J.E. Kooi and R. Babuska "*Inclined Quadrotor Landing using Deep Reinforcement Learning*", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.

Landing a quadrotor on an inclined surface is a challenging maneuver. The final state of any inclined landing trajectory is not an equilibrium, which precludes the use of most conventional control methods. We propose a deep reinforcement learning approach to design an autonomous landing controller for inclined surfaces. Using the proximal policy optimization (PPO) algorithm with sparse rewards and a tailored curriculum learning approach, an inclined landing policy can be trained in simulation in less than 90 minutes on a standard laptop. The policy then directly runs on a real Crazyflie 2.1 quadrotor and successfully performs real inclined landings in a flying arena. A single policy evaluation takes approximately 2.5 ms, which makes it suitable for a future embedded implementation on the quadrotor.

4.2.3 Future work

The presented method is a valuable step into bringing RL controllers into the real-world. However, we believe the method is experimental and far from suitable for general use. Hence, inclusion into the OpenDR toolkit is not desirable. Though, insights produced by this research on the transfer of simulated policies for quadcopters could potentially be incorporated in the agricultural use-case.

4.3 EAGERx

4.3.1 Introduction and objectives

Engine Agnostic Gym Environment with Reactive extension (EAGERx) is a toolkit that allows users to apply (deep) reinforcement learning for both simulated and real robots as well as combinations thereof. The toolkit serves as bridge between the popular reinforcement learning toolkit OpenAI Gym [8] and robots that can either be real or simulated. Thanks to communication based on reactive programming, EAGERx will allow users to speed up training in simulation while guaranteeing that actions and observations are synchronised.

OpenAI Gym is a toolkit for evaluating reinforcement learning algorithms in so-called Gym environments and is used for benchmarking (deep) reinforcement learning algorithms by the scientific community [39]. One of the benefits of OpenAI Gym is that users can easily evaluate state-of-the-art (deep) reinforcement learning algorithms for their Gym environments, thanks to the availability of algorithm implementations, such as Stable Baselines [18]. OpenAI Gym comes with a number of environments, including simulated robots, classic control tasks and Atari games.

Communication within EAGERx heavily depends on ReactiveX [2], which is a set of tools from the reactive programming paradigm. These tools allow one to perform operations on asynchronous data streams. Within EAGERx, these tools allow one to solve problems related to synchronization of actions and observations, such as ensuring that sensor measurements are updated after performing an action. This is vital in a reinforcement learning setting, since it is required that the transition of the environment's state is known in order to calculate the appropriate reward. Also, reactive programming allows EAGERx to run simulated training "as fast as possible", that is, all processing steps are performed as soon as the required data are available, since processes are event-based, rather than time-based. In a time-based setting problems can arise when control frequencies are increased, because some processes might be too computationally demanding to run at the specified frequency, while in a reactive setting, the nodes that are further downwards the data stream will wait until the data is available.

For reinforcement learning in robotics it is often highly desirable to train both in simulation and reality, because simulations allow to train faster than real-time and are safer than training with real robots. At the same time, real-world experience is required in many cases, because model inaccuracies of the simulator are exploited by reinforcement learning algorithms [22]. However, creating Gym environments for both real and simulated robots is currently a difficult and time-consuming task, because it is challenging to synchronize actions and observations, to communicate with both real and simulated robots and to add or interchange objects in environments. Therefore, we introduce EAGERx, which allows users to create engine agnostic Gym environments that can be used with different simulators, physics engines and with real robots. Also, by choosing an approach that is based on composition — instead of inheritance — adding robots, sensors and other objects to an environment will be reduced to a one-liner of code or a single click in the graphical user interface. The key functionalities that EAGERx will provide are:

1. User-friendly creation and modification of Gym environments for robot control tasks.
2. Integration with popular robot simulators Webots [27], PyBullet [14] and Gazebo [23].
3. Synchronization of actions and observations.
4. Switching between and/or combine simulated and real robots.
5. Engine agnostic processing of data streams, such as actions and observations.
6. The possibility to add procedures for resetting the environment after an episode.

In this section we will describe how the aforementioned functionalities will be provided.

(1) EAGERx allows users to easily create and modify Gym environments for robot control tasks. The toolkit is designed such that new environments can be created without the need to redefine objects, such as robots, sensors and actuators. Naturally, the objects need to be defined at least once. We aim to reduce the burden of defining objects and nodes as much as possible, by providing base classes and keeping the structure as homogeneous as possible. Eventually, we will stimulate the robotics community to add and share robot definitions to have an extensive number of robots and sensors supported by the EAGERx toolkit. Moreover, since ROS-based code is present in the backend of the toolkit, the toolkit can also be used by users without ROS experience. Nevertheless, the toolkit provides enough flexibility for experienced ROS users to exploit the possibilities of ROS. Furthermore, we are developing a graphical user interface to improve the intuitiveness of creating environments. In Figure 1 a screenshot is shown of the graphical user interface. This screenshot shows an example of an environment that can be created in EAGERx.

(2) EAGERx will provide integration with three simulators that are frequently used by the robotics and reinforcement learning communities, i.e. PyBullet, Gazebo and Webots. Also,

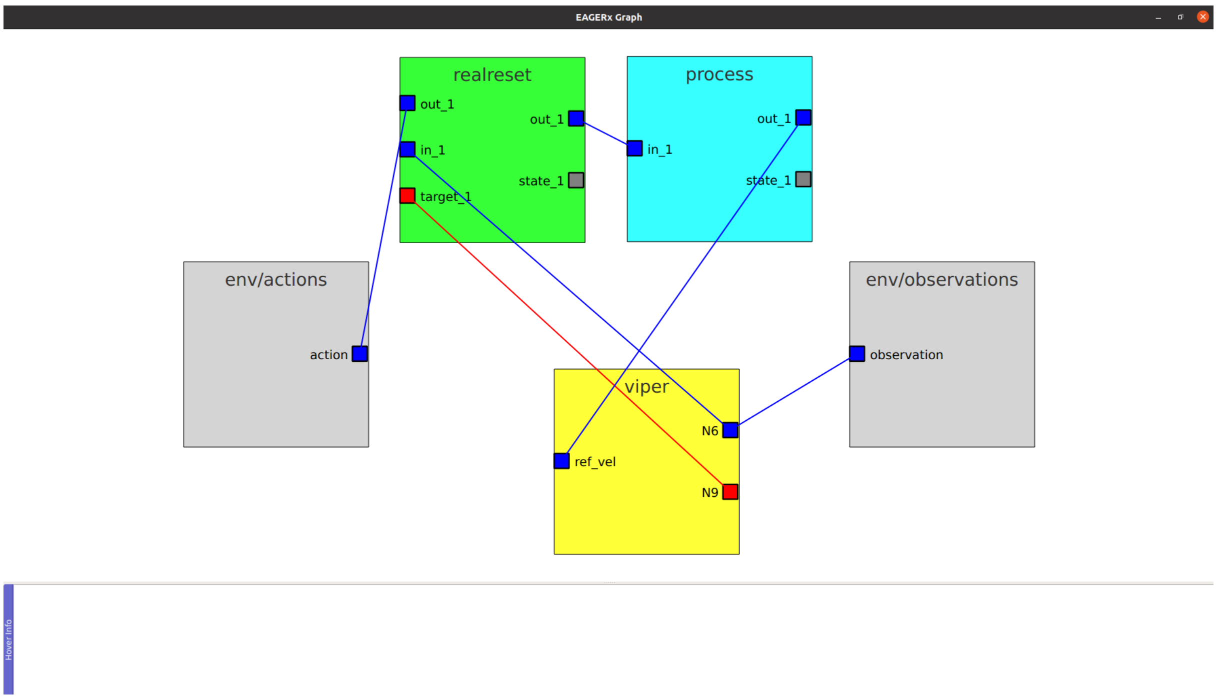


Figure 1: The graphical user interface will ease creating new EAGERx environments. New objects or processing nodes can be added to the environment with a single click and connections between nodes can be drawn. The graphical user interface is based on the PyQtGraph library [1].

users can train with different simulators in parallel and a base class is provided for creating a "bridge" for other simulators or physics engines.

(3) Synchronization of actions and observations is vital for effective reinforcement learning, because policy updates are based on state-action pairs and corresponding rewards in reinforcement learning. In order to pair actions and observations correctly, tools from the paradigm of reactive programming will be used.

(4) Thanks to the engine agnostic property of EAGERx environments, users can easily switch between real and simulated robots. Also, real and simulated robots could be trained in parallel, e.g., for simulator tuning.

(5) A base class will be provided for adding processing steps to actions, observations and other data streams. A similar base class will be provided for observations in the future. Processing of actions and observations can be useful, e.g., for checking whether actions are collision free, for obtaining the end-effector's position from joint states or for obtaining locations of objects that are detected in RGB images. Thanks to the implementation of the base class, users only have to implement the processing step and do not have to worry about communication and synchronization issues.

(6) A base class will be provided for reset procedures that will be executed at the end of episodes. This will allow users to automate their training procedures. This will also be provided for real-world training, but in that case there are limitations to the controllability of states.

4.3.2 Description of work performed so far

Initially, we have developed a version of EAGERx that is not reactive. In this version nodes would operate in a sequential fashion. While developing this toolkit, we realized that a reactive approach better suits the requirements of the toolkit. Therefore, we redeveloped the toolkit, where in the new version the communication is reactive. Simultaneously, while developing the new reactive version, we performed experiments with the initial version in order to validate the core functionalities of the toolkit. In these experiments, the goal was to move the end-effector of a six degrees of freedom manipulator to a cartesian position $\mathbf{r}_{\text{goal}}$ that was drawn from a uniform distribution at the beginning of the episode. The reward function R was a function of its current end-effector position $\mathbf{r}_{\text{ee}}$ and the time step i:

$$R(\mathbf{r}_{\text{ee}}, i) = \begin{cases} n_{\max} - i & \text{if } \|\mathbf{r}_{\text{ee}} - \mathbf{r}_{\text{goal}}\| \leq e \\ -n_{\max} + i & \text{if self-collision} \\ 0 & \text{else} \end{cases}, \tag{1}$$

where e is the goal tolerance (which was set to 0.05 m) and $n_{\max}$ the maximum number of steps per episode (which was set to 200). The episode was terminated (done) under the following conditions:

$$\text{done} = \begin{cases} \text{True} & \text{if } \|\mathbf{r}_{\text{ee}} - \mathbf{r}_{\text{goal}}\| < e \\ \text{True} & \text{if self-collision} \\ \text{True} & \text{if } i > n_{\max} \\ \text{False} & \text{else} \end{cases}. \tag{2}$$

The results of these experiments are shown in 2. Based on the plots that are shown in this figure we can conclude that training with the Soft Actor Critic (SAC) algorithm results in successful policies in both runs and therefore that the communication within EAGERx is functioning properly.

Furthermore, we have developed the aforementioned graphical user interface, which is shown in Figure 1. This will be in particular useful for complex environments, since in such cases defining the connections between different nodes and objects can become quite cumbersome. Also, it allows to easily load and save previously constructed environments in order to reuse or modify them.

4.3.3 Future work

We will continue the development of the graphical user interface in order to allow adding data stream converters, defining action/observation spaces and to incorporate a number of validity checks.

Also, we will continue to create definitions for new hardware and we will continue to document this process in order to make this as less of a burden as possible. Namely, we believe that the usefulness of the toolkit strongly depends on its usability.

Furthermore, we will continue to perform experiments with the toolkit. For example, we consider to perform experiments in which we fuse experience from different simulators and compare this to experience from a single simulator.

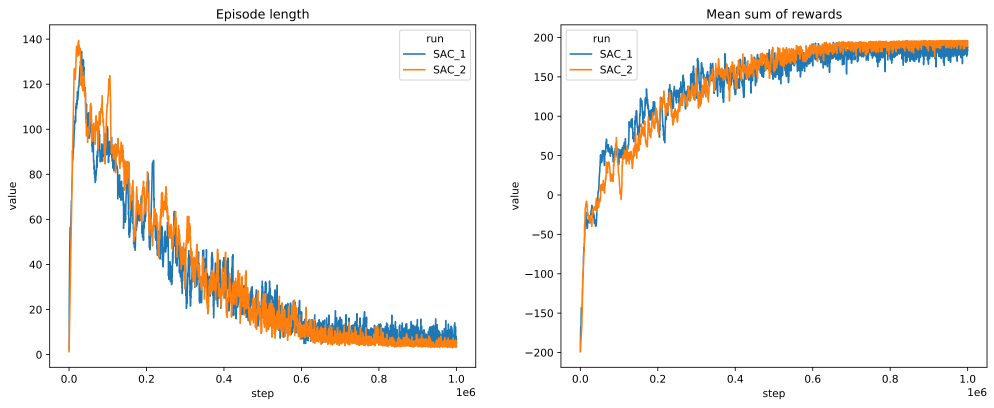


Figure 2: In order to test the core components of the toolkit, experiments were performed with a six degrees of freedom manipulator simulated in PyBullet. In these experiments, the goal was to move the end-effector to a Cartesian position that was drawn from a uniform distribution at the beginning of the episode. There were three termination conditions for the episode, i.e. self-collision, reaching the goal position or reaching the maximum number of steps (200). These figures show that the Soft Actor Critic (SAC) implementation from Stable Baselines [30] is able to solve this problem in an environment created using EAGERx (not yet using reactive programming). Interestingly, we found that the policy resulting from training in PyBullet seems to transfer to the real robot, but we do not yet have quantitative results for this.

4.3.4 Hyperparameter Tuning

4.3.5 Introduction and objectives

Tuning hyperparameters can be a tiresome — but also essential — part of developing deep learning tools. Since nearly all tools from the OpenDR toolkit involve hyperparameters that can influence the performance of the concerning tool significantly, we decided to choose for a holistic approach such that the tool is compatible with all learner classes from the OpenDR toolkit. Nowadays, popular hyperparameter tuning libraries exist, such as Optuna [3], Hyperopt [6] and Tune [26]. Rather than designing our own hyperparameter tuning library, we decided to design a tool that provides integration of the OpenDR toolkit with an existing hyperparameter tuning framework. We believe that this will improve the usability of the OpenDR toolkit. In the end, we chose to provide integration with the Optuna hyperparameter tuning framework, because of its versatility, visualisation tools, pruning capabilities and efficiency.

The objectives of the hyperparameter tuning tool are the following:

- Compatibility with all learner classes
- Easy setup of hyperparameter tuning
- Providing insights into influence of hyperparameters

4.3.6 Description of work performed so far

We developed a hyperparameter tuning utility tool within the OpenDR toolkit that provides integration with the Optuna hyperparameter tuning framework. This tool was designed with the aim to make it as holistic and user-friendly as possible. In order to use this tool, the user only needs to specify the learner class for which he or she would like to perform hyperparameter tuning and the arguments with which the learner should be initialized, trained and evaluated. After hyperparameter tuning, the visualisation tools from Optuna can be used to get insights into the hyperparameters and its importances. Figures 3 and 4 show examples of visualisations that can be obtained. A tutorial on how to perform hyperparameter tuning with this tool is available here. Documentation and tests have been created and a pull request is pending.

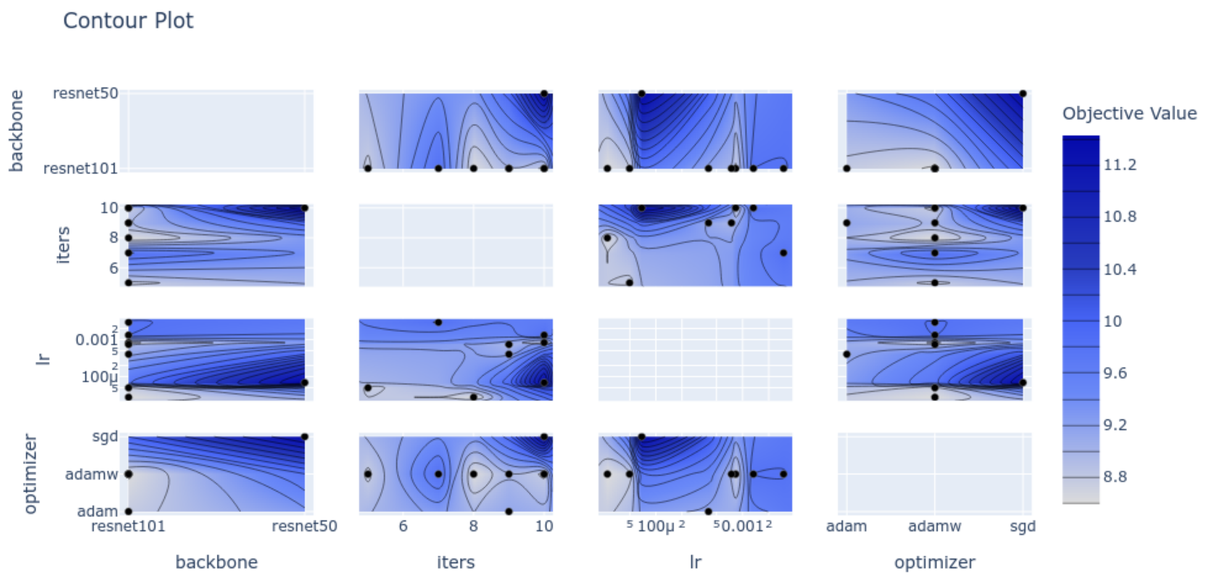


Figure 3: Optuna provides visualisation tools which can provide insights into the hyperparameter tuning process, e.g. hyperparameter contour plots.

4.3.7 Future work

The hyperparameter tuning tool will be evaluated on a practical hyperparameter tuning problem in order to validate its usefulness. Also, we will stimulate the developers of the OpenDR toolkit to provide definitions of the hyperparameters and their search spaces for each tool in order to alleviate this burden for users and improve the usability of the hyperparameter tuning tool.

4.4 Single demonstration grasping

4.4.1 Introduction and objectives

Collaborative robots (cobots) are getting more attention in the recent years as they bring safe human-robot interaction, easy and fast programming interface and wider ranges of applications. Despite the mentioned benefits, there are still works needed to be done in order to develop more effective robotic systems that are capable of efficiently collaborating with a human in a shared workspace. In this context, cobots are considered easy to program robots as they provide the so-called programming from demonstration which restricts the grasping to pre-defined locations. In order to mitigate this limitation, learning-based approached such as 2D/3D

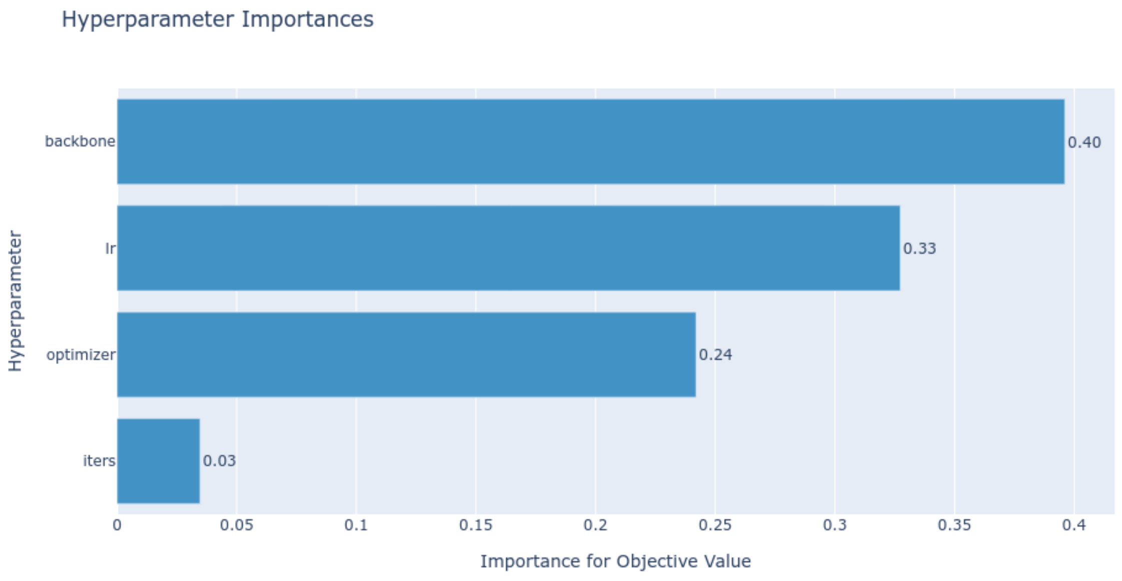


Figure 4: Optuna provides visualisation tools which can provide insights into the hyperparameter tuning process, e.g. hyperparameter importances plots.

object detection, end-to-end grasp detection etc. are being studied to a great extend where the objective is to obtain a model that learns to find suitable grasp poses for unseen objects and arbitrary locations. While the learning-based models usually requires a vast amount of training data and time to learn the grasp poses effectively, another approach is to train smaller non-generic networks that are unique to objects to directly output grasp poses or to predict other forms of outputs for example, object's location in 2/ 3-dimensional space, and find the grasp poses by post processing those outputs in combination with other vision methods. In this research work, a grasping pipeline is developed where the grasping in divided to four main steps of demonstration, augmentation, training and finally utilizing the outputs for a successful grasping action. We have evaluated the combination of 4 different vision-based methods for deriving the relative rotation of the object with respect to the reference/target frame alongside an object detection module. The most robust grasping pipeline was obtained using keypoint-rcnn network where the trained model is able to predict both object's location efficiently and several keypoints on the image that are used to calculate the relative rotation.

4.4.2 Description of work performed so far

The details of this work are presented in a draft article that is currently being prepared which is listed below, and can be found in Appendix F:

- A. Mehman Sefat, A. Angleraud, E. Rahtu and R. Pieters "*SingleDemoGrasp: Grasping from a single image demonstration*", in preparation, 2021.

In this work, a grasping pipeline is developed that utilizes keypoint-rcnn which is a variant of faster-rcnn to train a keypoint detector on a small augmented dataset which is generated by applying common augmentation techniques on the user's input images. During demonstration, the user is asked to draw center of grasp and a straight line on the object which is acting as the object's grasping orientation for parallel grippers. By performing inference on the objects, the planar 2D grasp poses will be translated to 3D space to to generate the final grasping motion.

4.4.3 Future work

The presented work demonstrates that the grasping pipeline can generate suitable data in order to train a light-weight model and successfully grasp objects with a robot. Even though the objects used are relevant for the agile production use case, a relatively small set of objects was evaluated. Future work will include more objects and object with higher complexity shape. In addition, current work considered objects to be on a planar surface, thereby reducing the object model complexity to 3D (2D position and 1D orientation). In future an object model with higher complexity will be considered enabling the grasping and manipulation of more complex shaped objects.

5 Human robot interaction

5.1 Human-Robot Collaboration by Commands

5.1.1 Introduction and objectives

Collaborative robots (cobots) are at increasing rate being deployed in industrial environments, sharing tasks and the work space with humans [38]. Tasks can be individually configured in a human-robot team setting, where the operator demonstrates task sequences and skills for the robot, and the robot repeats them [28]. This avoids having to go through a development phase, considerably speeding up integration time. Cobots are crucial for this, as they are small, light-weight and can be safely moved around by a human operator [25].

However, this programming of tasks is typically targeted only for independent robot motions, and task execution usually does not include human-robot interaction or physical collaboration. This implies that programming is still done offline, while the robot and the tasks are being prepared, and the actual execution phase is mostly autonomous execution of the robot. While applications can be found [32, 21, 15] that integrate coordinated actions (e.g., waiting for human input or trigger), still this is pre-programmed and planned to happen at certain specified occurrences. Coordination is thus planned in advance and both agents (i.e., human and robot) act as decided by a fixed protocol. If and when problems occur, or when changes need to be made in the collaboration, the work flow is disrupted and has to be restarted when problems get fixed or when changes are implemented. This limitation affects the natural collaboration and fluency between human and robot [19], as no spontaneous actions are allowed besides simply halting the robot and the action plan. While exceptions exist (see e.g., [15], which takes into account last-minute changes of task allocation), task plans are typically short, to avoid a large task plan network that is complex to model and track.

To allow more natural and fluent human-robot interaction, we believe collaboration between human and robot should be coordinated by the human, assisted by the robot and its knowledge and reasoning capabilities. At any given time during the collaboration, the human worker should be able to select suitable actions from the robot to assist the shared task. The robot verifies that the action is suitable and possible, based on its current state of the world and capabilities. Such knowledge is incorporated in a knowledge base that is updated at regular intervals by observations and human instructions. The selection of actions for the robot thus requires human commands to allow for intuitive instructions. Speech and text-based commands are most suitable as, similar to human-human communication [31], semantics can be included.

In this work, we present the developments to allow human coordination in shared human-robot collaborative tasks. The main contributions are:

- A knowledge-based system architecture that supports reasoning, planning and knowledge integration
- Shared task coordination by human commands, either by a graphical interface or by speech
- Industrially relevant use case scenarios that evaluate the approach

5.1.2 Description of work performed so far

The details of this work are found in the corresponding publication that is listed below, and can be found in Appendix E:

- [4] A. Angleraud, A. Mehman Sefat, M. Netzev and R. Pieters "*Coordinating Shared Tasks in Human-Robot Collaboration by Commands*", Frontiers in Robotics and AI, 8, 2021, *DOI:10.3389/frobt.2021.734548*.

In this work we explored the utilization of commands to coordinate a shared task between a human and a robot, in a shared work space. Based on a known set of higher-level actions (e.g., pick-and-placement, hand-over, kitting) and the commands that trigger them, both a speech-based and graphical command-based interface are developed to investigate its use. While speech-based interaction might be more intuitive for coordination, in industrial settings background sounds and noise might hinder its capabilities. The graphical command-based interface circumvents this, while still demonstrating the capabilities of coordination. The developed architecture follows a knowledge-based approach, where the actions available to the robot are checked at runtime whether they suit the task and the current state of the world. Experimental results on industrially relevant assembly, kitting and hand-over tasks in a laboratory setting demonstrate that graphical command-based and speech-based coordination with high-level commands is effective for collaboration between a human and a robot. Evaluation took into account metrics that assess the collaboration fluency between human and robot, such as human and robot idle time (H-IDL and R-IDL), functional delay (F-DEL) and concurrent activity (C-ACT).

5.1.3 Future work

Future work will combine computer vision and speech recognition for collaborative tasks, enabling human-robot collaboration that is more descriptive. For example, a human operator could command the robot to hand over a tool with a red handle from a table with multiple colored tools. The tools required for this (speech recognition and object detection) are taken from the OpenDR toolkit and will be tailored to the agile production use case.

6 Conclusions

This document presented the work performed on WP5. After a short introduction on the work done on the individual tasks, the document provided a detailed overview of the individual tasks, as detailed below.

Chapter 2 presented the status of the work performed for Task 5.1–Deep Planning. AU presented an end-to-end planner trained with DRL for local replanning in agricultural use-case. An agricultural simulation environment has been developed in Webots. The end-to-end planning algorithm is trained and tested in comprehensive simulations. The guidance of multiple UGVs is also demonstrated with a single UAV deployed with the end-to-end planner. The method is also deployed in real-world indoor environment successfully. The end-to-end planner outperforms a baseline implementation based on the artificial potential field method, which has a lower success rate, especially in cluttered obstacle settings. This shows that AgroRL has learned to make better long-term decisions. The importance of a high-level reward in DRL training is also verified by providing a reward for successfully finishing an episode to the agent where the agent shows an 18% higher success rate. One downside of the method is that it is not sufficient to deploy continuously on an onboard computer, such as NVIDIA Jetson TX2. So, the method is implemented discretely where a new position reference is provided once in a while. To decrease computational complexity, a VAE-based representation of the depth image is utilized in training. However, the performance cannot match the proposed method.

Chapter 3 detailed the status of the work performed for Task 5.2–Deep Navigation. ALU-FR introduced a novel approach for mobile manipulation, which allows to very easily define and execute novel mobile manipulation tasks. The approach combines reinforcement learning with inverse kinematics to decompose the long horizon problem and introduces a novel, dense reward for training. The tool has been successfully integrated into the OpenDR toolkit. This work is currently being extended to cluttered and human-centered environments as well as to dynamic obstacles. We furthermore developed an approach to audio-visual navigation that increases the complexity of the task and improves generalization to unheard sounds by a large margin.

Chapter 4 highlighted the work performed for Task 5.3–Deep Action and Control. First, TUD introduced the work performed on model-based latent planning. If an extension of the proposed method can achieve sufficient performance with manipulators, it is intended to be added to the toolkit. Else, TUD will focus on more promising research directions. Second, TUD presented a method for learning a quadcopter inclined landing policy in simulation that directly transferred to the real world. Valuable insights on sim2real transfer of quadcopter policies can be drawn from this research that could prove valuable for the quadcopters used in the agricultural use-case. TUD presents the development of the EAGERx toolkit that will bridge the gap between OpenAI Gym (the standard evaluation tool for reinforcement learning) and robotics. The toolkit facilitates the creation of complex Gym environments for robot control tasks by exploiting reactive programming tools and a relying on composition, rather than inheritance. Development of this toolkit will continue as well as evaluation of the toolkit. Also, TUD developed a hyperparameter tuning tool for which a holistic approach was chosen in order to be compatible with all learner classes from the OpenDR toolkit. This tool integrates the functionalities of the Optuna [3] hyperparameter tuning framework in the OpenDR toolkit. The tool will be evaluated on a practical problem in order to validate its usefulness. TAU introduced a grasping pipeline that generates the required training data from a single object image demonstration and utilizes this data to learn and detect suitable grasp poses from a single image observation. The success of the method is demonstrated by real grasping experiments with high grasp success rate (>90%).

Finally, Chapter 5 highlighted the work performed for Task 5.4–Human Robot Interaction. The chapter covered the following topics. TAU developed a collaborative scenario between human and robot, which is inspired by the Agile Production use case. Commands are utilized as interaction modality to coordinate the collaboration. Both graphical commands (GUI) and speech are utilized with contributions of WP3 (speech recognition). The results demonstrate its use with respect to metrics that are typical in agile production, i.e., human and robot idle time (H-IDL and R-IDL), concurrent activity (C-ACT) and functional delay (F-DEL).

References

[1] PyQtGraph: Scientific Graphics and GUI Library for Python. `https://www.pyqtgraph.org/`. Accessed: 2021-11-29.

[2] ReactiveX: An API for asynchronous programming with observable streams. `http://reactivex.io/`. Accessed: 2021-11-29.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[4] A. Angleraud, A. Mehman Sefat, M. Netzev, and R. Pieters. Coordinating shared tasks in human-robot collaboration by commands. *Frontiers in Robotics and AI*, 8:332, 2021.

[5] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, et al. An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962. IEEE, 2012.

[6] J. Bergstra, D. Yamins, D. D. Cox, et al. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, page 20. Citeseer, 2013.

[7] K. Blomqvist, M. Breyer, A. Cramariuc, J. Förster, M. Grinvald, F. Tschopp, J. J. Chung, L. Ott, J. Nieto, and R. Siegwart. Go fetch: Mobile manipulation in unstructured environments. *arXiv preprint arXiv:2004.00899*, 2020.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] F. Burget, A. Hornung, and M. Bennewitz. Whole-body motion planning for manipulation of articulated objects. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1656–1662, 2013.

[10] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niebner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *2017 International Conference on 3D Vision (3DV)*, pages 667–676. IEEE, 2017. Matterport3D dataset licence available at: `http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf`.

[11] D. S. Chaplot, D. P. Gandhi, A. Gupta, and R. R. Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[12] C. Chen, U. Jain, C. Schissler, S. V. A. Gari, Z. Al-Halah, V. K. Ithapu, P. Robinson, and K. Grauman. Soundspaces: Audio-visual navigation in 3d environments. In *European Conference on Computer Vision*, pages 17–36. Springer, 2020. Sound dataset licence available at: `https://github.com/facebookresearch/sound-spaces/blob/main/LICENSE`.

[13] C. Chen, S. Majumder, Z. Al-Halah, R. Gao, S. K. Ramakrishnan, and K. Grauman. Learning to set waypoints for audio-visual navigation. In *International Conference on Learning Representations*, 2020.

[14] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[15] K. Darvish, E. Simetti, F. Mastrogiovanni, and G. Casalino. A hierarchical architecture for human–robot cooperation processes. *IEEE Transactions on Robotics*, 37(2):567–586, 2021.

[16] M. Fortin, P. Voss, C. Lord, M. Lassonde, J. Pruessner, D. Saint-Amour, C. Rainville, and F. Lepore. Wayfinding in the blind: larger hippocampal volume and supranormal spatial navigation. *Brain*, 131(11):2995–3005, 2008.

[17] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *IEEE Conference Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.

[18] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[19] G. Hoffman. Evaluating fluency in human–robot collaboration. *IEEE Transactions on Human-Machine Systems*, 49(3):209–218, 2019.

[20] D. Honerkamp, T. Welschehold, and A. Valada. Learning kinematic feasibility for mobile manipulation through deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):6289–6296, 2021.

[21] L. Johannsmeier and S. Haddadin. A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes. *IEEE Robotics and Automation Letters*, 2(1):41–48, 2017.

[22] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[23] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[24] J. E. Kooi and R. Babuska. Inclined quadrotor landing using deep reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[25] S. Kumar, C. Savur, and F. Sahin. Survey of human-robot collaboration in industrial settings: Awareness, intelligence, and compliance. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):280–297, 2021.

[26] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[27] O. Michel. Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.

[28] U. E. Ogenyi, J. Liu, C. Yang, Z. Ju, and H. Liu. Physical human-robot collaboration: Robotic systems, learning methods, collaborative strategies, sensors, and actuators. *IEEE transactions on cybernetics*, 51(4):1888–1901, 2021.

[29] F. Paus, P. Kaiser, N. Vahrenkamp, and T. Asfour. A combined approach for robot placement and coverage path planning for mobile manipulation. *International Conference on Intelligent Robots and Systems*, 2017.

[30] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[31] A. Rocci and L. d. Saussure. *Verbal communication*. De Gruyter, 2016.

[32] B. Sadrfaridpour and Y. Wang. Collaborative assembly in hybrid manufacturing cells: an integrated framework for human–robot interaction. *IEEE Transactions on Automation Science and Engineering*, 15(3):1178–1192, 2017.

[33] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A Platform for Embodied AI Research. In *International Conference on Computer Vision*, 2019. HabitatLab licence available at: `https://github.com/facebookresearch/habitat-lab/blob/v0.1.6/LICENSE`.

[34] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5, 2010.

[35] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. Replica dataset licence available at: `https://github.com/facebookresearch/Replica-Dataset/blob/main/LICENSE`.

[36] J. Stückler, M. Schwarz, and S. Behnke. Mobile manipulation, tool use, and intuitive interaction for cognitive service robot cosero. *Frontiers in Robotics and AI*, 3:58, 2016.

[37] B. van der Heijden, L. Ferranti, J. Kober, and R. Babuska. Deepkoco: Efficient latent planning with a task-relevant koopman representation. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[38] V. Villani, F. Pini, F. Leali, and C. Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018.

[39] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[40] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. Ddppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *International Conference on Learning Representations*, 2019.

A

September 27 - October 1, 2021. Prague, Czech Republic

DeepKoCo: Efficient latent planning with a task-relevant Koopman representation

Bas van der Heijden[1], Laura Ferranti[1], Jens Kober[1], Robert Babuška[1]

***Abstract*— This paper presents DeepKoCo, a novel model-based agent that learns a latent Koopman representation from images. This representation allows DeepKoCo to plan efficiently using linear control methods, such as linear model predictive control. Compared to traditional agents, DeepKoCo learns task-relevant dynamics, thanks to the use of a tailored lossy autoencoder network that allows DeepKoCo to learn latent dynamics that reconstruct and predict only observed costs, rather than all observed dynamics. As our results show, DeepKoCo achieves a similar final performance as traditional model-free methods on complex control tasks, while being considerably more robust to distractor dynamics, making the proposed agent more amenable for real-life applications.**

***Index Terms*— Model-based reinforcement learning, Koopman theory, model-predictive control**

I. INTRODUCTION

From self-driving cars to vision-based robotic manipulation, emerging technologies are characterized by visual measurements of strongly nonlinear physical systems. Unlike in highly controlled lab environments where any measured change is likely relevant, cameras in real-world settings are notorious for mainly capturing task-irrelevant information, such as, the movement of other robots outside of a manipulator's workspace or cloud movements captured by the cameras of self-driving cars.

While Deep Reinforcement Learning (DRL) algorithms can learn to perform various tasks using raw images, they will require an enormous number of trials. Prior methods mitigate this by encoding the raw images into a lower-dimensional representation that allows for faster learning. However, these methods can be easily distracted by irrelevant dynamics [1]. This motivates data-driven methodologies that learn low-dimensional latent dynamics that are task-relevant and useful for control.

In the learning of latent dynamics for control, there is a trade-off between having an accurate dynamic model and one that is suitable for control. On one hand, latent dynamic models based on neural networks (NN) can provide accurate predictions over long horizons. On the other hand, their inherent nonlinearity renders them incompatible with efficient planning algorithms. Alternatively, one can choose to approximate the latent dynamics with a more restricted function approximation class to favor the use of efficient planning algorithms. In this respect, a promising strategy is represented by the Koopman framework [2]. Loosely speaking, this framework allows one to map observations with nonlinear dynamics to a latent space where the *global dynamics* of the autonomous system are approximately linear (Koopman representation). This enables the use of powerful linear optimal control techniques in the latent space [2].

While the Koopman framework is promising, existing methods have fundamental limitations that must be addressed to fully exploit the benefits of this method for control applications. First, methods that identify Koopman representations from data were designed for prediction and estimation. These methods were later adapted for control. These adaptations, however, lead to limiting assumptions on the underlying dynamics, such as assuming the Koopman representation to be linear in the states and actions [3], [4], [5], [6], [7]. Second, these methods are *task agnostic*, that is, the models represent all dynamics they observe, whether they are relevant to the task or not. This focuses the majority of their model capacity on potentially task-irrelevant dynamics.

Therefore, we introduce *Deep Koopman Control* (DeepKoCo), that is, a model-based agent that learns a latent Koopman representation from raw pixel images and achieves its goal through planning in this latent space. The representation is *(i)* robust to task-irrelevant dynamics and *(ii)* compatible with efficient planning algorithms. We propose a lossy autoencoder network that reconstructs and predicts observed costs, rather than all observed dynamics, which leads to a representation that is task-relevant. The latent-dynamics model can represent continuously differentiable nonlinear systems and does not require knowledge of the underlying environment dynamics or cost function. We demonstrate the success of our approach on two continuous control tasks and show that our method is more robust to irrelevant dynamics than state-of-the-art approaches, that is, DeepMDP [8] and Deep Bisimulation for Control (DBC) [1].

II. RELATED WORK

1) Koopman control: Koopman theory has been used to control various nonlinear systems with linear control techniques, both in simulation [2], [9], [6] and in real-world robotic applications [4], [5]. Herein, [10], [2], [4] used a linear quadratic regulator (LQR), while [9], [3], [5], [6], [7] applied linear model predictive control (MPC). [9], [3], [5], [7] used data-driven methods that were derived from the Extended Dynamic Mode Decomposition (EDMD) [11] to find the Koopman representation. In contrast, [4], [2], [10] require prior knowledge of the system dynamics to hand-craft parts of the lifting function. Similar to [6], we rely on deep

*This work was supported by the European Union's H2020 project Open Deep Learning Toolkit for Robotics (OpenDR) under grant agreement No 871449.

[1]Cognitive Robotics at the Faculty of 3mE, Delft University of Technology, The Netherlands. d.s.vanderheijden@tudelft.nl

learning to derive the Koopman representation for control. However, we do not assume the Koopman representation to be (bi-)linear in the states and actions and we show how our representation can be used to control systems that violate this assumption. Compared to existing methods, we propose an agent that learns the representation online in a reinforcement learning setting using high-dimensional observations that contain irrelevant dynamics.

2) Latent planning: Extensive work has been conducted to learn latent dynamics from images and use them to plan suitable actions [12], [13], [14]. [14] proposes a model-based agent that uses NNs for the latent dynamics and cost model. To find suitable action sequences, however, their method requires a significant computational budget to evaluate many candidate sequences. Alternatively, [12], [13] propose locally linear dynamic models, which allowed them to efficiently plan for actions using LQR. However, their cost function was defined in the latent space and required observations of the goal to be available. In contrast to our approach, all aforementioned methods are trained towards full observation reconstruction, which focuses the majority of their model capacity on potentially task-irrelevant dynamics.

3) Relevant representation learning: [8], [1] filter task-irrelevant dynamics by minimizing an auxiliary bisimulation loss. Similar to our approach, they propose learning latent dynamics and predicting costs. Their method, however, is limited to minimizing a single-step prediction loss, while we incorporate multi-step predictions. This optimizes our model towards accurate long-term predictions. [15], [16] also proposed training a dynamics model towards predicting the future sum of costs given an action sequence. However, their method focused on discrete control variables, while we focus on continuous ones.

III. PRELIMINARIES

This section briefly introduces the Koopman framework for autonomous and controlled nonlinear systems. A detailed description can be found in [2]. This framework is fundamental to the design of our latent model and control strategy.

A. Koopman eigenfunctions for autonomous systems

Consider the following autonomous nonlinear system $\dot{\boldsymbol{o}} = F(\boldsymbol{o})$, where the observations $\boldsymbol{o} \in \mathbb{R}^N$ evolve according to the smooth continuous-time dynamics $F(\boldsymbol{o})$. For such a system, there exists a lifting function $g(\cdot) : \mathbb{R}^N \to \mathbb{R}^n$ that maps the observations to a latent space where the dynamics are linear, that is,

$$\frac{\mathrm{d}}{\mathrm{d}t} g(\boldsymbol{o}) = \mathcal{K} \circ g(\boldsymbol{o}), \tag{1}$$

where $\mathcal{K}$ is the infinitesimal operator generator of Koopman operators K. In theory, K is infinite dimensional (i.e., $n \to \infty$), but a finite-dimensional matrix representation can be obtained by restricting it to an invariant subspace. Any set of eigenfunctions of the Koopman operator spans such a subspace. Identifying these eigenfunctions [9], [17] provides a set of intrinsic coordinates that enable global linear representations of the underlying nonlinear system. A Koopman eigenfunction satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t} \phi(\boldsymbol{o}) = K\phi(\boldsymbol{o}) = \lambda\phi(\boldsymbol{o}), \tag{2}$$

where $\lambda \in \mathbb{C}$ is the continuous-time eigenvalue corresponding to eigenfunction $\phi(\boldsymbol{o})$.

B. Koopman eigenfunctions for controlled systems

For controlled nonlinear system with action $\boldsymbol{a} \in \mathbb{R}^m$ and smooth continuous-time dynamics $\dot{\boldsymbol{o}} = \tilde{F}(\boldsymbol{o}, \boldsymbol{a})$, we follow the procedure in [2]. Given the eigenfunction $\phi(\boldsymbol{o}, \boldsymbol{a})$ augmented with $\boldsymbol{a}$ for the controlled system, we can take its time derivative and apply the chain rule with respect to $\boldsymbol{o}$ and $\boldsymbol{a}$, leading to

$$\frac{\mathrm{d}}{\mathrm{d}t} \phi(\boldsymbol{o}, \boldsymbol{a}) = \underbrace{\nabla_{\boldsymbol{o}} \phi(\boldsymbol{o}, \boldsymbol{a}) \tilde{F}(\boldsymbol{o}, \boldsymbol{a})}_{\lambda\phi(\boldsymbol{o},\boldsymbol{a})} + \nabla_{\boldsymbol{a}}\, \phi(\boldsymbol{o}, \boldsymbol{a}) \dot{\boldsymbol{a}}, \tag{3}$$

where λ is now the eigenvalue that corresponds to eigenfunction $\phi(\boldsymbol{o}, \boldsymbol{a})$. Since $\dot{\boldsymbol{a}}$ can be chosen arbitrarily, we could set it to zero and instead interpret each action as a parameter of the Koopman eigenfunctions. Thus, for any given choice of parameter $\boldsymbol{a}$ the standard relationship in (2) is recovered in the presence of actions. A local approximation of the Koopman representation is obtained when $\dot{\boldsymbol{a}}$ is nonzero.

C. Identifying Koopman eigenfunctions from data

To facilitate eigenfunction identification with discrete data, (3) can be discretized with a procedure similar to [17]. The eigenvalues $\lambda_{\pm} = \mu \pm \mathrm{i}\,\omega$ are used to parametrize block-diagonal $\Lambda = \mathrm{diag}(J^{[1]}, J^{[2]}, ..., J^{[P]}) \in \mathbb{R}^{2P \times 2P}$. For all P pairs of complex eigenvalues, the discrete-time operator Λ has a Jordan real block of the form

$$J(\mu, \omega) = e^{\mu\Delta t} \begin{bmatrix} \cos(\omega\Delta t) & -\sin(\omega\Delta t) \\ \sin(\omega\Delta t) & \cos(\omega\Delta t) \end{bmatrix}, \tag{4}$$

with sampling time Δt. The "forward Euler method" provides a discrete approximation of the control matrix, so that (3) can be discretized as

$$\varphi(\boldsymbol{o}_{k+1}, \boldsymbol{a}_{k+1}) = \Lambda\varphi(\boldsymbol{o}_k, \boldsymbol{a}_k) + \underbrace{\nabla_{\boldsymbol{a}_k} \varphi(\boldsymbol{o}_k, \boldsymbol{a}_k)}_{B_{\varphi_k}} \underbrace{\dot{\boldsymbol{a}}_k \Delta t}_{\Delta \boldsymbol{a}_k}. \tag{5}$$

Herein, the stacked vector $\varphi = (\phi^{[1]}, \phi^{[2]}, ..., \phi^{[P]})$ comprises a set of P eigenfunctions with $\phi^{[j]} \in \mathbb{R}^2$ associated with complex eigenvalue pair $\lambda_{\pm}^{[j]}$ and Jordan block $J^{[j]}$. Subscript k corresponds to discretized snapshots in time. If we view the action increment $\Delta\boldsymbol{a}_k = \boldsymbol{a}_{k+1} - \boldsymbol{a}_k$ in (5) as the controlled input instead, we obtain a discrete *control-affine* Koopman eigenfunction formulation with *linear autonomous* dynamics for the original *non-control-affine nonlinear* system. In the next section, we show that (5) plays a central role in our latent model.

IV. LEARNING TASK-RELEVANT KOOPMAN EIGENFUNCTIONS

For efficient planning in the latent space, we propose to learn a latent dynamics model that uses Koopman eigenfunctions as its latent state. This section describes this model and how the Koopman eigenfunctions can be identified robustly, that is, in a way that the identified eigenfunctions remain unaffected by task-irrelevant dynamics that are expected to contaminate the observations.

A. Koopman latent model

We propose a *lossy* autoencoder that builds on the deep autoencoder in [17]. Compared to [17], our autoencoder enables control. To train the latent model, we provide the training objective that is to be minimized given a buffer D that contains observed sequences $\{t_k\}_{k=0}^{T}$ of a Markov decision process with tuples $t_k = (\boldsymbol{o}_k, \boldsymbol{a}_k, \Delta\boldsymbol{a}_k, c_k)$, where c_k are observed scalar costs. The proposed latent model is illustrated in Fig. 1 with more details below on the individual components of the architecture.

1) Encoder: The encoder φ is the approximate eigenfunction that maps an observation-action pair $(\boldsymbol{o}_k, \boldsymbol{a}_k)$ to the latent state $\boldsymbol{s}_k$. The encoder φ is parametrized by a neural network, defined as

$$\boldsymbol{s}_k = \varphi(\boldsymbol{o}_k, \boldsymbol{a}_k). \tag{6}$$

2) Latent dynamics: The latent state $\boldsymbol{s}_k$ approximates a Koopman eigenfunction, so the autonomous time evolution in the latent space is linear and dictated by Λ. Note here that $\boldsymbol{a}_k$ is part of the *augmented* latent state and we view the action increment $\Delta\boldsymbol{a}_k$ as the controlled variable that is determined by the policy. This leads to the dynamics model in (7), which we derived from (5). The Koopman operator Λ is parametrized by P complex conjugate eigenvalue pairs $\lambda_{\pm}^{[j]}$. We do not assume the latent dynamics to be linear in the control. Instead, the influence of $\Delta\boldsymbol{a}_k$ on the latent state varies and depends on the partial derivative of the encoder with respect to the action, i.e., the state-dependent matrix $B_{\varphi_k} = \nabla_{\boldsymbol{a}_k}\varphi(\boldsymbol{o}_k, \boldsymbol{a}_k) \in \mathbb{R}^{2P\times m}$.

$$\begin{bmatrix}\boldsymbol{s}_{k+1}\\ \boldsymbol{a}_{k+1}\end{bmatrix} = \begin{bmatrix}\Lambda & 0\\ 0 & I\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_k\\ \boldsymbol{a}_k\end{bmatrix} + \begin{bmatrix}B_{\varphi_k}\\ I\end{bmatrix}\Delta\boldsymbol{a}_k. \tag{7}$$

3) Cost model: The environment contains a cost function that produces a scalar cost observation c_k at every time-step. For planning in the latent space, we require a cost model $\hat{c}_k$ as a function of the latent state. This cost approximates the observed cost (i.e., $\hat{c}_k \approx c_k$). We adopt a latent state-dependent quadratic cost model to facilitate the use of fast planning algorithms (Section V). The entries of $C_{\boldsymbol{s}_k} \in \mathbb{R}^{1\times 2P}$ are determined by a function $\psi(\boldsymbol{s}_k)$ that is parametrized by a neural network. The weights of ψ are initially unknown and must be learned together with the rest of the latent model. We assume that the cost of applying action $\boldsymbol{a}_k$ is known *a priori* and defined by matrix R. This leads to the cost model

$$\hat{c}_k = ||C_{\boldsymbol{s}_k}\boldsymbol{s}_k||_2^2 + \boldsymbol{a}_k^{\mathsf{T}} R \boldsymbol{a}_k. \tag{8}$$

4) Policy: The action increment $\Delta\boldsymbol{a}_k$ is the controlled variable that is sampled from a probability distribution π, conditioned on the augmented latent state (i.e., $\Delta\boldsymbol{a}_k \sim \pi(\Delta\boldsymbol{a}_k|\boldsymbol{s}_k, \boldsymbol{a}_k)$). Even though the model is deterministic, we define the policy to be stochastic to allow for stochastic exploration. The policy will be specified further in Section V.

$$\Delta\boldsymbol{a}_k \sim \pi(\Delta\boldsymbol{a}_k|\boldsymbol{s}_k, \boldsymbol{a}_k) \tag{9}$$

5) Decoder: After learning the latent model we intend to plan over it, which involves a multi-step prediction. Given only the encoder (6), dynamics model (7), policy (9) and the current $(\boldsymbol{o}_k, \boldsymbol{a}_k)$-pair, we would be limited to single-step predictions of $\boldsymbol{s}_{k+1}$ at run-time, because multi-step predictions $\boldsymbol{s}_{k+i}$ with $i > 1$ would require knowledge of future observations $\boldsymbol{o}_{k+i}$ to evaluate $B_{\varphi_{k+i}}$. Therefore, to make multi-step predictions, we introduce a decoder φ^{-1} (parametrized by a NN) in (10) that uses predicted latent states $\boldsymbol{s}_{k+i}$ to construct pseudo-observations $\hat{\boldsymbol{o}}_{k+i}$ that produce the same partial derivative as the true observation (i.e., $\nabla_{\boldsymbol{a}_k}\varphi(\varphi^{-1}(\boldsymbol{s}_k), \boldsymbol{a}_k) \approx \nabla_{\boldsymbol{a}_k}\varphi(\boldsymbol{o}_k, \boldsymbol{a}_k)$). Future values $\boldsymbol{a}_{k+i}$ do not pose a problem, because they can be inferred from the policy (9) and dynamics model (7).

$$\hat{\boldsymbol{o}}_k = \varphi^{-1}(\boldsymbol{s}_k). \tag{10}$$

6) Image processor: When the observations are raw pixel images $\boldsymbol{p}_k$, not all relevant information can be inferred from a single observation. To restore the Markov property, we pass the last d consecutive pixel images through a convolutional neural network Ω in (11), stack the output into a single vector, and consider that to be the observation $\boldsymbol{o}_k$ instead. In that case, the observed sequences consist of tuples $t_k = (\boldsymbol{p}_k, ..., \boldsymbol{p}_{k-d+1}, \boldsymbol{a}_k, \Delta\boldsymbol{a}_k, c_k)$.

$$\boldsymbol{o}_k = \Omega(\boldsymbol{p}_k, ..., \boldsymbol{p}_{k-d+1}), \tag{11}$$

B. Learning the latent model

Our latent model should have linear dynamics and be predictive of observed costs. These two high-level requirements lead to the following three losses which are minimized during training.

1) Linear dynamics: To ensure that the latent state is a valid Koopman eigenfunction, we regularize the time evolution in the latent space to be linear by using the following loss,

$$\text{Linear loss:} \quad \mathcal{L}_{\text{lin}} = \frac{1}{T}\sum_{k=0}^{T-1} ||\varphi(\boldsymbol{o}_{k+1}, \boldsymbol{a}_{k+1}) - \boldsymbol{s}_{k+1}||_{\text{MSE}}, \tag{12}$$

where $\boldsymbol{s}_{k+1}$ is obtained by rolling out a latent trajectory as illustrated in Fig. 1.

2) Cost prediction: We want the latent representation to contain all necessary information to solve the task. If we would naively apply an autoencoder that predicts future observations, we focus the majority of the model capacity on potentially task-irrelevant dynamics contained in the observations. To learn a latent representation that *only* encodes relevant information, we propose to use a lossy autoencoder

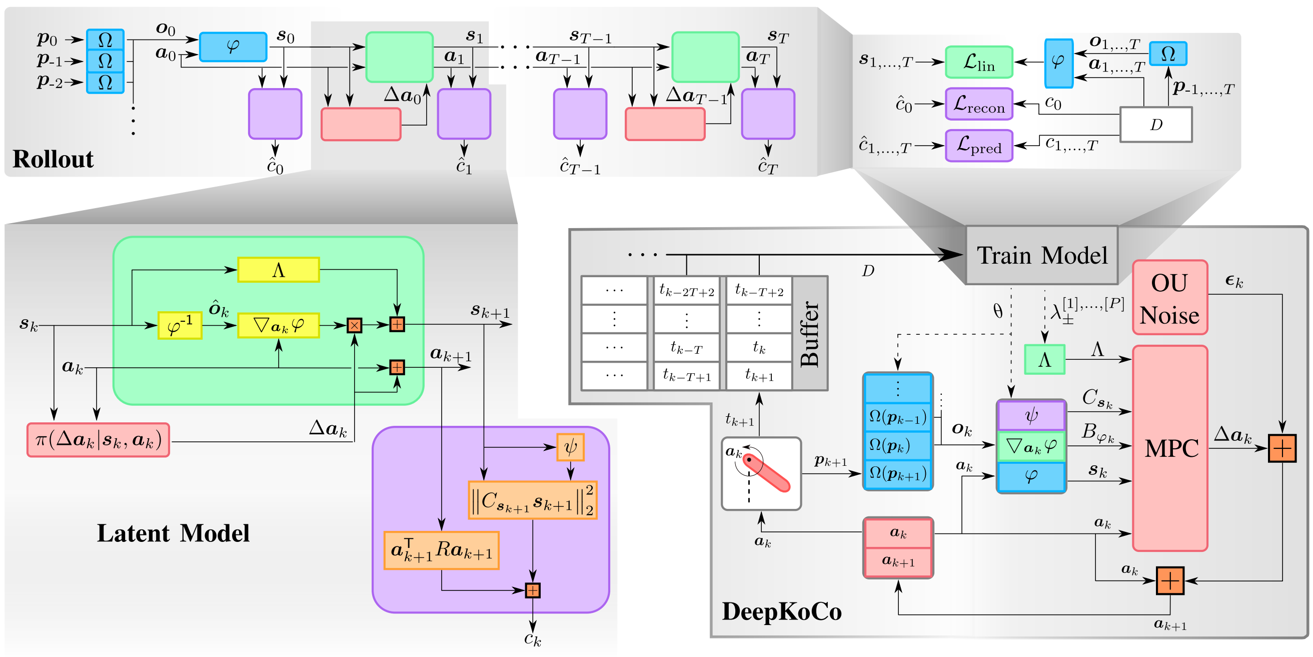


Fig. 1: **Latent Model** The proposed network architecture of the *latent model*, consisting of the dynamic model, cost model, and policy (depicted in green, purple, and red, respectively). **Rollout** A multi-step ahead prediction with the *latent model*. Note that we only encode an observation at the first time-step (blue boxes), after which we remain in the latent space. **DeepKoCo** The training procedure that corresponds to Algorithm 1.

that is predictive of current and future costs instead. Such a representation would allow an agent to predict the cost evolution of various action sequences and choose the sequence that minimizes the predicted cumulative cost, which is essentially equivalent to solving the task. Because we only penalize inaccurate cost predictions, the encoder is not incentivized to encode task-irrelevant dynamics into the latent representation as they are not predictive of the cost. This leads to the task-relevant identification of the lifting function φ. Cost prediction accuracy is achieved by using the following two losses,

$$\text{Reconstruction loss:} \quad \mathcal{L}_{\text{recon}} = \|c_0 - \hat{c}_0\|_{\text{MSE}} \tag{13}$$

$$\text{Prediction loss:} \quad \mathcal{L}_{\text{pred}} = \frac{1}{T}\sum_{k=1}^{T}\|c_k - \hat{c}_k\|_{\text{MSE}} \tag{14}$$

3) Training objective: We minimize the losses in (12), (13), and (14), corresponding to linear dynamics regularization and cost prediction, together with an $L2$-regularization loss $\mathcal{L}_{\text{reg}}$ on the trainable variables (excluding neural network biases). This leads to the following training objective,

$$\min_{\theta,\lambda_{\pm}^{[1],\dots,[P]}} \mathcal{L}_{\text{lin}} + \alpha_1(\mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{pred}}) + \alpha_2\mathcal{L}_{\text{reg}}, \tag{15}$$

where θ is the collection of all the trainable variables that parametrize the encoder φ, decoder φ^{-1}, cost model ψ, and convolutional network Ω (in case of image observations). Weights α_1, α_2 are hyperparameters. The model is trained using the Adam optimizer [18] with learning rate $\tilde{\alpha}$, on batches of B sequences $\{t_k\}_{k=0}^{T}$ for E epochs.

V. DEEP KOOPMAN CONTROL

This section introduces the agent that uses the Koopman latent model to find the action sequence that minimizes the predicted cumulative cost. We use linear model-predictive control (LMPC) to allow the agent to adapt its plan based on new observations, meaning the agent re-plans at each step. Re-planning at each time-step can be computationally costly. In the following, we explain how to exploit and adapt our latent model and cost model to formulate a sparse and convex MPC problem that can be solved efficiently online.

The planning algorithm should achieve competitive performance, while only using a limited amount of computational resources. This motivates choosing Koopman eigenfunctions as the latent state, because the *autonomous* dynamics are linear. The dynamics are affine in the controlled variable $\Delta\boldsymbol{a}_k$ that is multiplied in the definition of the state space by B_{φ_k}, which depends on the latent state. Similarly, $C_{\boldsymbol{s}_k}$ requires the evaluation of the nonlinear function $\psi(\boldsymbol{s}_k)$. There exist methods that can be applied in this setting, such as the State-Dependent Ricatti Equation (SDRE) method [19]. While the SDRE requires less complexity compared to sample-based nonlinear MPC (e.g. CEM [20]), it remains computationally demanding as it also requires the derivative of ψ with respect to s_k at every step of the planning horizon.

Our goal is to reduce the online complexity of our planning strategy, while also dealing with input constraints. Hence, we trade-off some prediction accuracy (due to the mismatch between the latent model and the MPC prediction model) to simplify the online planning strategy by using linear MPC.

We propose to evaluate the state-dependent matrices $C_{\boldsymbol{s}_0}$ and B_{φ_0} at time-step $k=0$ (obtained from our latent model) and keep them both fixed for the rest of the LMPC horizon. This assumes that the variation of B_{φ_k} and $C_{\boldsymbol{s}_k}$ is limited over the prediction horizon (compared to (7) and (8)). Nevertheless, thanks to this simplification we can rely on LMPC for planning that can be solved efficiently. Specifically, once we evaluate $\boldsymbol{s}_0$, $C_{\boldsymbol{s}_0}$, and B_{φ_0}, the computational cost of solving the MPC problem in the *dense form* [21] scales linearly with the latent state dimension due to the diagonal structure of Λ. As Section VI details, this simplification allows our method to achieve competitive final performance, while only requiring a single evaluation of the NNs Ω, φ, and ψ. This significantly decreases the computational cost at run-time compared to sample-based nonlinear MPC (e.g., CEM [20]) that would require many evaluations of the NNs at every time-step. In contrast to LQR, LMPC can explicitly deal with actuator saturation by incorporating constraints on $\boldsymbol{a}$. The proposed planning strategy based on LMPC is defined as follows:

$$\min_{\Delta\boldsymbol{a}^{[0],\dots,[H-1]}} \sum_{k=1}^{H} ||C_{\boldsymbol{s}_0}\boldsymbol{s}_k||_2^2 + \boldsymbol{a}_k^{\intercal} R \boldsymbol{a}_k + \Delta\boldsymbol{a}_k^{\intercal}\tilde{R}\Delta\boldsymbol{a}_k, \quad (16)$$

$$\text{s.t.} \quad \begin{bmatrix}\boldsymbol{s}_{k+1}\\ \boldsymbol{a}_{k+1}\end{bmatrix} = \begin{bmatrix}\Lambda & 0\\ 0 & I\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_k\\ \boldsymbol{a}_k\end{bmatrix} + \begin{bmatrix}B_{\varphi_0}\\ I\end{bmatrix}\Delta\boldsymbol{a}_k,$$

$$\boldsymbol{a}^{\min} \leq \boldsymbol{a}_k \leq \boldsymbol{a}^{\max}, \text{ for } k=1,\dots,H,$$

where H is the prediction horizon. Positive-definite matrix $\tilde{R}$ penalizes the use of $\Delta\boldsymbol{a}_k$ and is required to make the problem well-conditioned. Its use does introduce a discrepancy between the approximate cost model (8) and the cumulative cost ultimately minimized by the agent (16). Therefore, the elements in $\tilde{R}$ are kept as low as possible.

To align the representation learning objective (15) with the linear MPC objective (16), we also fix the state-dependent terms $C_{\boldsymbol{s}_0}$ and B_{φ_0} at time-step $k=0$ in the evaluation of the cost prediction loss (14) and linear loss (12). Note that this does not mean that $C_{\boldsymbol{s}_k}$ and B_{φ_k} are constant in the latent model (7), (8). The matrices remain state-dependent, but their variation is limited over the sequence length T. In general, we choose the sequence length to be equal to the prediction horizon. Hence, we learn a representation that provides local linear models that are particularly accurate around $\boldsymbol{s}_k$ in the direction of the (goal-directed) trajectories gathered during training.

To gather a rich set of episodes to learn the Koopman latent model, we add colored noise to the actions commanded by the agent's linear MPC policy, that is, $\boldsymbol{a}_{k+1} = \boldsymbol{a}_k + \Delta\boldsymbol{a}_k + \boldsymbol{\epsilon}_k$. This adds a stochastic exploration component to the policy. We use an Ornstein-Uhlenbeck (OU) process to generate the additive colored noise with decay rate λ^{ou}. The variance $\sigma^{2,\text{ou}}$ is linearly annealed from $\sigma^{2,\text{ou}}_{\text{init}} \rightarrow 0$ over $1,\dots,N^{\text{ou}}$ episodes, that is, after N^{ou} episodes the policy becomes deterministic.

An overview of the proposed method is shown in Algorithm 1. First, we initialize all model parameters. Then, we construct the Koopman operator Λ with (4) and gather N episodes of experience. Each time-step, we process the image observations with (11), evaluate the latent model (6), (7), and (8), and use it to find $\Delta\boldsymbol{a}_k$ with (16). Noise is added to the action increment before it is applied to the environment. We fill the experience buffer D with N episodes, split into sequences of length T, and train on them for E epochs with (15). This is repeated until convergence.

Algorithm 1: Deep Koopman Control (DeepKoCo)

```
Input: Model parameters: $P, d$
       Policy parameters: $\zeta = \{H, R, \tilde{R}\}$
       Noise parameters: $\lambda^{\text{ou}}, \sigma^{2,\text{ou}}_{\text{init}}, N^{\text{ou}}$
       Train parameters: $N, L, T, E, B$
       Optimization parameters: $\xi = \{\alpha_1, \alpha_2, \tilde{\alpha}\}$
Output: Eigenvalues $\lambda_{\pm}^{[1],\dots,[P]}$
        Trained networks $\varphi, \varphi^{-1}, \psi, \Omega$
1  $\theta, \lambda_{\pm}^{[1],\dots,[P]} \leftarrow$ InitializeModel$(P, R)$
2  while not converged do
3      $\Lambda \leftarrow$ GetKoopmanOperator$(\lambda_{\pm}^{[1],\dots,[P]})$
4      for episode $l = 1,\dots,N$ do
5          $\boldsymbol{p}_0, \dots, \boldsymbol{p}_{1-d} \leftarrow$ ResetEnvironment()
6          $\boldsymbol{a}_0 \leftarrow 0$
7          for time-step $k = 0,\dots,L$ do
8              $\boldsymbol{o}_k \leftarrow$ ProcessImages$(\boldsymbol{p}_k, \dots, \boldsymbol{p}_{k-d+1}, \theta)$
9              $\boldsymbol{s}_k, B_{\varphi_k}, C_{\boldsymbol{s}_k} \leftarrow$ LatentModel$(\boldsymbol{o}_k, \boldsymbol{a}_k, \theta)$
10             $\Delta\boldsymbol{a}_k \leftarrow$ LMPC$(\boldsymbol{s}_k, \boldsymbol{a}_k, B_{\varphi_k}, C_{\boldsymbol{s}_k}, \Lambda, \zeta)$
11             $\boldsymbol{a}_{k+1} \leftarrow \boldsymbol{a}_k + \Delta\boldsymbol{a}_k +$ Noise$(\lambda^{\text{ou}}, \sigma^{2,\text{ou}})$
12             $\boldsymbol{p}_{k+1}, c_k \leftarrow$ ApplyAction$(\boldsymbol{a}_k)$
13         $D \leftarrow D \cup$ CreateSequences$(T, \{t_k\}_{k=0}^{L})$
14     $\theta, \lambda_{\pm}^{[1],\dots,[P]} \leftarrow$
           TrainModel$(D, \theta, \lambda_{\pm}^{[1],\dots,[P]}, \xi, E, B)$
```

VI. RESULTS

We evaluate *DeepKoCo* on two continuous control tasks, namely OpenAI's pendulum swing-up task and a manipulator task. The manipulator task is similar to OpenAI's reacher task where the two joints of a two-link arm are torque controlled and the Euclidean distance between the arm's end-point and a target must be minimized. However, we increase the difficulty by allowing the target to move at a constant angular velocity and radius around the arm's center-joint. Given that the angular velocity and radius vary randomly over episodes, the manipulator must learn to track arbitrary circular trajectories at different speeds. The dynamics for the manipulator can be formulated as $\boldsymbol{x}_{k+1} = F(\boldsymbol{x}_k) + B(\boldsymbol{x}_k)\boldsymbol{a}_k$, where $\boldsymbol{x}$ is the original nonlinear state. Such dynamics do not necessarily admit a Koopman representation that is (bi-)linear in the states and actions, as is often assumed in literature [6], [4].

To investigate the effect of distractor dynamics, we test each task in two different scenarios. In the first scenario, only relevant dynamics are observed, while in the second one we purposely contaminate the observations with distractor dynamics. The agent is either provided with a concatenated state observation containing the state measurements of both the relevant system and distractors (while not knowing which

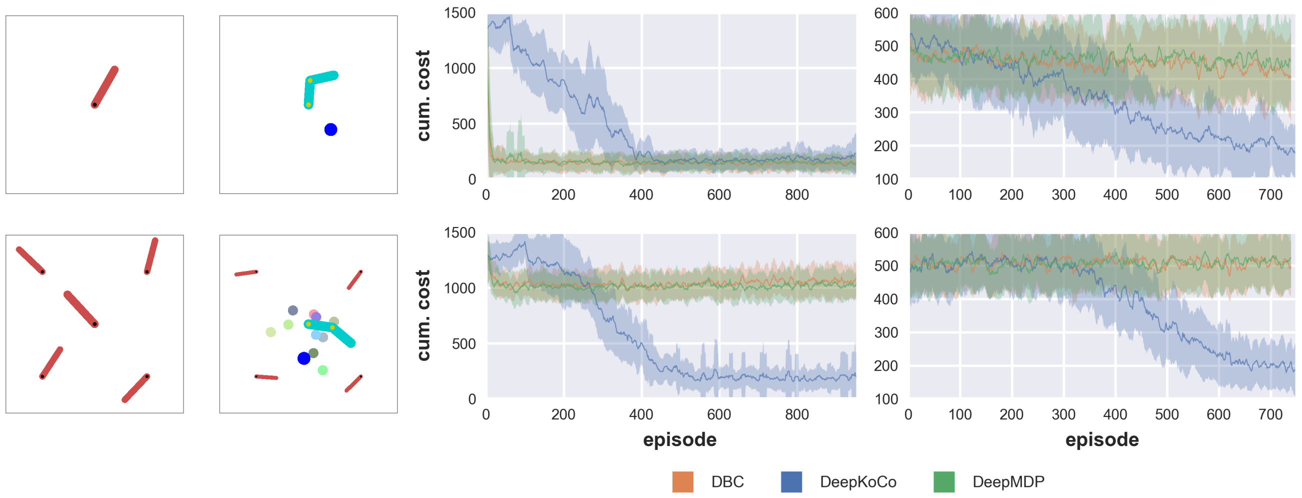


Fig. 2: **Left** Typical setups for the two tasks in a clean scenario (first row) and distractor scenario (second row). In all setups, the center system is controlled by the agent. In the manipulator task, the moving target is a blue ball. **Right** Learning curves when using state observations. The grid-location of each figure corresponds to the grid-location of each setup on the left. The mean cumulative cost over the last 10 episodes (line) with one standard deviation (shaded area) over 5 random seed runs are shown.

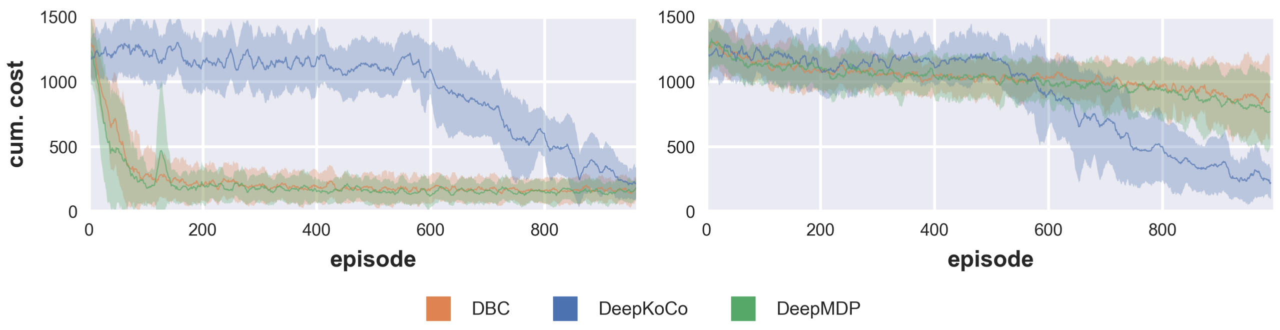


Fig. 3: Learning curves when using images as observations in the pendulum task. The mean cumulative cost over the last 10 episodes (line) with one standard deviation (shaded area) over 5 random seed runs are shown. **Left** Clean scenario. **Right** Distractor scenario.

states are the relevant ones) or image observations with all systems in-frame (refer to Fig. 2 for the setup). The state observation dimension from the clean to the distractor scenario increases from 3 to 15 for the pendulum and from 10 to 50 for the manipulator. A video of the simulations using DeepKoCo accompanies the paper [22].

1) Baselines: We compare with two baselines that both combine model-free reinforcement learning with an auxiliary bisimulation loss to be robust against task-irrelevant dynamics: *(i)* Deep Bisimulation for Control (DBC) [1], and *(ii)* DeepMDP [8]. In case of state observations, we replace their convolutional encoder, with our fully connected encoder.

2) Hyperparameters: We use the same set of hyperparameters throughout all experiments, except for the number of complex eigenvalue pairs P, that is, $P = 10$ and $P = 30$ in the swing-up and manipulator task, respectively to cope with the complexity of the scenarios. As policy parameters, we use $H = 15, R = 0.001, \tilde{R} = 0.01, \lambda^{\text{OU}} = 0.85, \sigma^{2,\text{ou}}_{\text{init}} = 0.85$. Initially, we fill the experience buffer D with $N = 90$ episodes, split into sequences of length $T = 15$, and train on them for $E = 100$ epochs. Then, we continuously add the sequences of $N = 20$ episodes to the buffer and train on the complete buffer for $E = 3$ epochs. As optimization parameters, we use $\alpha_1 = 10, \alpha_2 = 10^{-14}, \tilde{\alpha} = 0.001$. In case of image observations, we stack the last $d = 3$ images, downsample them to $3 \times 64 \times 64$ pixels before passing them through the convolutional NN defined in [23]. The networks $\varphi, \varphi^{-1}, \psi$ are 2-layered fully connected NNs with 90, 90, and 70 units per layer, respectively. The layers use ReLU activation and are followed by a linear layer. The number of complex eigenvalue pairs P, planning horizon H, and action increment cost $\tilde{R}$ are the most important parameters to tune.

3) Clean scenario: Concerning the pendulum task, the baselines converge more quickly to the final performance compared to DeepKoCo, as the top-left graph of Fig. 2 shows. Nevertheless, we do consistently achieve a similar final performance. The slower convergence can be explained by the added noise, required for exploration, that is only fully annealed after 400 episodes. We believe the convergence rate can be significantly improved by performing a parameter search together with a faster annealing rate, but we do not expect to be able to match the baselines in this ideal scenario. Note that, despite the apparent *simplicity* of the application scenario, finding an accurate Koopman representation for the pendulum system is challenging, because it exhibits a continuous eigenvalue spectrum and has multiple (unstable) fixed points [17]. Concerning the manipulator task, both baselines were not able to solve the manipulator task with a moving target as the top-right graph of Fig. 2 shows, while they were able to learn in case the target was fixed (results omitted due to space limitations). This shows that learning to track arbitrary circular references is significantly harder than regulating towards a fixed goal. Despite the increased difficulty, DeepKoCo learns to track the moving target. Finally, note that the manipulator task shows that the proposed method can deal with a multi-dimensional action-space and non-quadratic cost functions, that is, the Euclidean norm (the square root of an inner product).

4) Distractor scenario: In the more realistic scenario, both baselines fail to learn anything. In contrast, our approach is able to reach the same final performance as in the clean scenario, in a comparable amount of episodes, as the bottom row of Fig. 2 shows. This result can be explained by noticing that our multi-step cost prediction (in our loss function (14)) provides a stronger learning signal for the agent to distinguish relevant from irrelevant dynamics. For the manipulator task, there is a tracking error caused by the trade-off of using fixed B_φ and C_s along the MPC prediction horizon for efficiency. While our latent model presented in Section IV supports state-dependent matrices, we decided to keep them fixed in the control design for efficiency.

5) Image observations: Fig. 3 shows the results for the pendulum task when images are used instead of state observations. In both clean and distractor scenarios, our approach is able to reach a similar final performance compared to using state observations. As expected, the baselines struggle to learn in the distractor scenario. This supports our statement that our approach learns a task-relevant Koopman representation from high-dimensional observations. We plan to test the manipulator task with images both in simulation and in real-world experiments.

VII. CONCLUSIONS AND FUTURE WORK

We presented a model-based agent that uses the Koopman framework to learn a latent Koopman representation from images. DeepKoCo can find Koopman representations that *(i)* enable efficient linear control, *(ii)* are robust to distractor dynamics. Thanks to these features, DeepKoCo outperforms the baselines (two state-of-the-art model-free agents) in the presence of distractions. As part of our future work, we will extend our deterministic latent model with stochastic components to deal with partial observability and aleatoric uncertainty. Furthermore, we will extend our cost model to deal with sparse rewards, as they are often easier to provide.

REFERENCES

[1] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine, "Learning invariant representations for reinforcement learning without reconstruction," *arXiv preprint arXiv:2006.10742*, 2020.

[2] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of Koopman eigenfunctions for control," in *APS Division of Fluid Dynamics Meeting Abstracts*, ser. APS Meeting Abstracts, Nov. 2017, p. M27.006.

[3] H. Arbabi, M. Korda, and I. Mezić, "A data-driven koopman model predictive control framework for nonlinear partial differential equations," *IEEE Conference on Decision and Control (CDC)*, pp. 6409–6414, 2018.

[4] G. Mamakoukas, M. Castano, X. Tan, and T. Murphey, "Local koopman operators for data-driven control of robotic systems," in *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[5] D. Bruder, B. Gillespie, C. D. Remy, and R. Vasudevan, "Modeling and control of soft robots using the koopman operator and model predictive control," in *Proceedings of Robotics: Science and Systems (RSS)*, 2019.

[6] Y. Li, H. He, J. Wu, D. Katabi, and A. Torralba, "Learning compositional koopman operators for model-based control," in *International Conference on Learning Representations (ICLR)*, 2020.

[7] M. Korda and I. Mezic, "Optimal construction of koopman eigenfunctions for prediction and control," *IEEE Transactions on Automatic Control*, 2020.

[8] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare, "DeepMDP: Learning continuous latent space models for representation learning," *36th International Conference on Machine Learning (ICML)*, pp. 3802–3826, 2019.

[9] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, 2018.

[10] S. L. Brunton, B. W. Brunton, J. L. Proctor, and J. N. Kutz, "Koopman invariant subspaces and finite linear representations of nonlinear dynamical systems for control," *PloS one*, vol. 11, no. 2, 2016.

[11] M. O. Williams, I. G. Kevrekidis, and C. W. Rowley, "A Data–Driven Approximation of the Koopman Operator: Extending Dynamic Mode Decomposition," *Journal of Nonlinear Science*, vol. 25, no. 6, pp. 1307–1346, 2015.

[12] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Advances in neural information processing systems (NIPS)*, 2015, pp. 2746–2754.

[13] E. Banijamali, R. Shu, H. Bui, A. Ghodsi *et al.*, "Robust locally-linear controllable embedding," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 1751–1759.

[14] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 2555–2565.

[15] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 6118–6128.

[16] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *arXiv preprint arXiv:1911.08265*, pp. 1–21, 2019.

[17] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, no. 1, pp. 1–10, dec 2018.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] I. Chang and J. Bentsman, "Constrained discrete-time state-dependent riccati equation technique: A model predictive control approach," in *52nd IEEE Conference on Decision and Control (CDC)*. IEEE, 2013, pp. 5125–5130.

[20] I. Szita and A. Lörincz, "Learning tetris using the noisy cross-entropy method," *Neural computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[21] J. M. Maciejowski, *Predictive control: with constraints*. Pearson education, 2002.

[22] B. van der Heijden, L. Ferranti, J. Kober, and R. Babuška, "Supplementary video material of DeepKoCo simulations," https://youtu.be/75lOuQyHBmQ, July 2021.

[23] D. Ha and J. Schmidhuber, "World Models," *arXiv preprint arXiv:1803.10122*, 2018.

B

September 27 - October 1, 2021. Prague, Czech Republic

Inclined Quadrotor Landing using Deep Reinforcement Learning

Jacob E. Kooi[1] and Robert Babuška[2]

Abstract— **Landing a quadrotor on an inclined surface is a challenging maneuver. The final state of any inclined landing trajectory is not an equilibrium, which precludes the use of most conventional control methods. We propose a deep reinforcement learning approach to design an autonomous landing controller for inclined surfaces. Using the proximal policy optimization (PPO) algorithm with sparse rewards and a tailored curriculum learning approach, an inclined landing policy can be trained in simulation in less than 90 minutes on a standard laptop. The policy then directly runs on a real Crazyflie 2.1 quadrotor and successfully performs real inclined landings in a flying arena. A single policy evaluation takes approximately 2.5 ms, which makes it suitable for a future embedded implementation on the quadrotor.**

I. Introduction

Modern quadrotors are agile and can perform complex tasks in difficult-to-reach places. Quadrotor flight and maneuvers are commonly controlled by proportional integral derivative (PID) control or model predictive control (MPC). Although these methods are adequate for set-point or trajectory tracking, they fall short when it comes to more complicated maneuvers that exceed the linearization range or require long prediction horizons. One such maneuver is the landing on an inclined surface, which is relevant for applications like delivery, maintenance, or surveillance. To facilitate a safe inclined landing, the final attitude of the quadrotor must match the slope of the landing platform. The final state of the landing trajectory is not an equilibrium, which presents a challenge for the control design. Owing to the under-actuated nature of the system, the landing trajectory can be long and complex, with an initial motion away from the landing location. This complicates the use of standard control methods like MPC with a fixed prediction horizon and quadratic cost function.

Recent advances in deep reinforcement learning (DRL) with continuous action spaces have made this approach suitable also for quadrotor control [1], [2], [3], [4], including landing controllers [5], [6], [7], [8], [9], [10], [11]. However, no results have yet been reported for inclined landing. In this paper, we develop a DRL approach to the inclined landing problem and validate it in simulations and real lab experiments with the Crazyflie 2.1 Nano-UAV. To the best of our knowledge, this is the first deep-learning-based controller for inclined landing applied to a real quadcopter. More specifically, our contributions are:

- We develop two fast Gym-based [12] simulation environments for the Crazyflie 2.1 Nano-UAV.[3] One three-dimensional environment can be used with any compatible DRL algorithm to train set-point tracking. The other two-dimensional environment, restricted to the vertical xz-plane, can be used with an on-policy algorithm to train the inclined landing. The resulting policies adequately transfer to the real Crazyflie.
- Building upon the state-of-the-art model-free proximal policy optimization (PPO) algorithm [13], we propose a powerful curriculum learning [14] approach to facilitate convergence when using sparse rewards, without the need for applying a multi-goal setting like in hindsight experience replay [15] or iterated supervised learning [16].
- We test the trained policy network in simulation and then deploy it to the real Crazyflie quadrotor to demonstrate the actual inclined landing in an indoor flying arena.[4]

The remainder of the paper is structured as follows. We first give an overview of the related work in Section II. The dynamic quadrotor model used for simulation and training is described in Section III. Next, Section IV presents the DRL simulation framework that is used to train inclined landing and set-point tracking. Section V describes the simulation and lab setup and presents the experimental results. Section VI contains a discussion of the results and in Section VII, the conclusions and limitations of this work are given, along with proposals for future work.

II. Related Work

Deep reinforcement learning methods have been applied to a variety of quadrotor control problems, including hovering [3], attitude control [2], set-point tracking, and disturbance recovery [1], [4]. Specifically for landing, a deep neural network was employed to learn higher-order interactions to stabilize the near-ground behavior of a nonlinear quadrotor controller [5]. A deep Q-learning network (DQN) was used to detect a marker symbol and perform a landing by using a downward-facing low-resolution camera [7], [8], [9]. The work in [8] considers platform inclination, but only in the context of more involved visual recognition, while the landing is still horizontal. Least-squares policy iteration (LSPI) was employed to autonomously land on a marker [11] and the

[1]Jacob E. Kooi is with the Department of Cognitive Robotics and Delft Center for Systems and Control, Delft University of Technology, 2628 CD Delft, The Netherlands `jacobkooi92@gmail.com`

[2]Robert Babuška with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD Delft, The Netherlands and with the Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Czech Republic `r.babuska@tudelft.nl`

[3]https://github.com/Jacobkooi/InclinedDroneLander.git

[4]https://youtu.be/pJ6vVs0BsB8

deep deterministic policy gradient (DDPG) algorithm [17] was used to navigate a descending quadrotor to land on a moving platform [10]. Finally, the work in [6] involved a convolutional neural network to estimate the heading angle to aid UAV landing in the case of sensor failure. However, none of the approaches considered inclined landing and none of the methods developed can be directly applied to this problem.

Inclined landing has been the topic of several works outside the deep learning control literature. A nonlinear hybrid controller was proposed in [18]. A trajectory-tracking controller first guides the quadrotor above the landing platform and then switches to an attitude-tracking controller to ensure that the attitude of the quadrotor adjusts to the slope of the landing platform upon touchdown. This is an ad hoc local strategy, incapable of generating optimal landing trajectories from arbitrary initial conditions. Besides, no real-time control experiments have been reported in this paper. The method proposed in [19] features a nonlinear MPC to land a quadrotor on a moving inclined surface. Real-time experimental results were reported, showing a successful landing. The limitations of MPC are its computational complexity and the difficulty of parameter tuning, especially of the prediction horizon, which needs to be long for some of the landing trajectories, making the method unsuitable for embedded implementation on the quadrotor. The approach developed in [20] relies on splitting up the problem in the generation of dynamically feasible trajectories and their subsequent trajectory tracking. Perching on slopes of up to 90 degrees has been demonstrated in lab experiments. To keep the problem tractable, the authors break the desired trajectory down into segments with a maximum duration of one second. The overall approach is more complex than the nonlinear feedback policy approach pursued in this paper.

III. Simulation Model

The dynamic model of the Crazyflie 2.1 Nano-UAV is formed by the equations of motion (EOM). We divide them into the Newton-Euler equations, which govern the axial accelerations, and an approximation of the body attitude control loops. The command input vector u to the Crazyflie's onboard controller is defined as

$$u = \begin{bmatrix}\Theta_c & \phi_c & \theta_c & \dot{\psi}_c\end{bmatrix}^T. \tag{1}$$

Here, Θ_c is the commanded pulse-width modulation (PWM) signal representing the total thrust, ϕ_c and θ_c are the commanded roll and pitch angles, respectively, and $\dot{\psi}_c$ is the commanded yaw rate [21]. These inputs are bounded by

$$\begin{aligned} u_{\min} &= \begin{bmatrix}10000 & -30^\circ & -30^\circ & -200^\circ/s\end{bmatrix}^T, \\ u_{\max} &= \begin{bmatrix}60000 & 30^\circ & 30^\circ & 200^\circ/s\end{bmatrix}^T. \end{aligned} \tag{2}$$

A. Newton-Euler Equations

The quadrotor is modeled as a rigid body, with the axial accelerations in the inertial frame $\begin{bmatrix}x & y & z\end{bmatrix}^T$:

$$\begin{bmatrix} m\ddot{x} \\ m\ddot{y} \\ m\ddot{z} \end{bmatrix} = \mathbf{R}\left(\begin{bmatrix}0\\0\\F_t\end{bmatrix} + F_a\right) + \begin{bmatrix}0\\0\\-mg\end{bmatrix} \tag{3}$$

with m the quadrotor's mass, $\mathbf{R}$ the rotation matrix from the body frame to the inertial frame, F_t the total thrust force and F_a the drag force. The rotation matrix corresponding to the coordinate frame representation in Fig. 1 is

$$\mathbf{R} = \begin{bmatrix} c\psi c\theta - s\phi s\psi s\theta & -c\phi s\psi & c\psi s\theta + c\theta s\phi s\psi \\ c\theta s\psi + c\psi s\phi s\theta & c\phi c\theta & s\psi s\theta + c\psi c\theta s\phi \\ -c\phi s\theta & s\phi & c\phi c\theta \end{bmatrix} \tag{4}$$

where c is a cosine, s is a sine, and ϕ, θ and ψ are the roll, pitch and yaw angles, respectively.

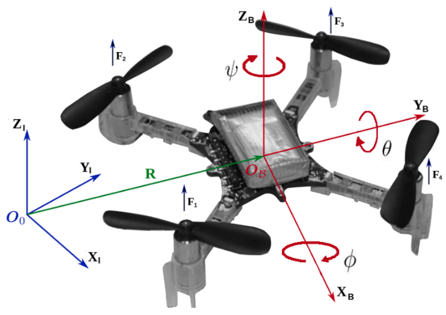


Fig. 1. The quadrotor coordinate system used throughout this paper. Subscripts B and I represent the body and inertial frame, respectively, and F_i is the thrust due to rotor i. Adopted from [22].

The relation between the commanded PWM Θ_c in (1) and F_t in (3) is modelled by using the discrete-time transfer function found in [23] for an individual motor i:

$$\frac{F_i(z)}{\Theta_{c,i}(z)} = \frac{7.2345374 \cdot 10^{-8}}{1 - 0.9695404 z^{-1}} \qquad (500Hz). \tag{5}$$

Since the Crazyflie's onboard controller only takes a single PWM signal for all four motors, we assume that $F_t \approx 4F_i$ with $\Theta_c \approx \Theta_{c,i}$. Multiplying (5) by four and converting it to continuous-time gives

$$\begin{bmatrix}\dot{\Omega} \\ F_t\end{bmatrix} = \begin{bmatrix} -15.467 \\ 1.425 \cdot 10^{-4}\end{bmatrix}\Omega + \begin{bmatrix} 1 \\ 2.894 \cdot 10^{-7}\end{bmatrix}\Theta_c \tag{6}$$

where Ω is an unmeasured state used for simulation purposes only. The drag force F_a in (3) is expressed as [24]

$$F_a = \boldsymbol{K}_a \omega_\Sigma v \tag{7}$$

with ω_Σ the sum of the rotor velocities, v the body-frame velocity vector and $\boldsymbol{K}_a$ a diagonal matrix of drag constants estimated in [23]. Because ω_Σ is not known during simulation, we approximate it from F_t with additional conversion formula's given in [23].

B. Body Attitude Control Loops

The body attitude rates are modelled by equations that approximate the dynamics of the attitude control loops [25]:

$$\begin{aligned} \dot{\phi} &= \frac{1}{\tau_\phi}(k_\phi \phi_c - \phi), \\ \dot{\theta} &= \frac{1}{\tau_\theta}(k_\theta \theta_c - \theta), \\ \dot{\psi} &= \dot{\psi}_c. \end{aligned} \tag{8}$$

Here τ_ϕ, τ_θ and k_ϕ, k_θ are the time and gain constants for roll and pitch, respectively. The yaw rate is assumed to instantaneously track the desired yaw rate, which is a reasonable assumption since the yaw has no effect on the quadrotor's position [25]. Because the closed-loop dynamics are unknown, the parameters k_θ, k_ϕ, τ_θ and τ_ϕ need to be identified. Given the quadrotor's symmetry, k_θ and k_ϕ as well as τ_θ and τ_ϕ are assumed equal. These parameters are estimated by fitting the data gathered by a motion-capture system to the equations (8). We conducted 20 experiments using square and sine waves ranging from zero to thirty degrees, which gave an average fit of 85.3% using Matlab's `nlgreyest` function, with the resulting parameters $k_\phi = k_\theta = 1.1094$ and $\tau_\phi = \tau_\theta = 0.1838\,\text{s}$.

To simulate the quadrotor, the model equations (3) and (8) are integrated by using the fourth-order Runge Kutta (RK4) method. The step size is fixed and equal to the sampling period $T_s = 0.02\,\text{s}$.

IV. Training Deep Reinforcement Learning Policies

To train the inclined landing, the quadrotor model of Section III is used as a simulation environment for model-free DRL. Additionally, to navigate the quadrotor, we train set-point tracking in the same fashion. Both policy networks map quadrotor states to the desired control input in (1), which makes them directly applicable to the real Crazyflie.

A. Preliminaries

The learning controller (agent) interacts with the model (environment) through trials. The environment's state space is denoted by $\mathcal{S}$ and a specific value of the state at time step k by s_k. The agent applies an action $a_k \in \mathcal{A}$ and subsequently receives a reward $r_k \in \mathbb{R}$, after which it observes the next state s_{k+1}. The action a_k is chosen by following a stochastic policy $\pi_k(a|s)$ or a deterministic policy $\mu_k(s)$. This policy can be optimized in many different ways. Most techniques maximize the discounted return $\eta(\pi_\phi) = \mathbb{E}_\tau[\sum_{t=0}^{T} \gamma^t r(s_k, a_k)]$, with τ a trajectory following the policy π_ϕ and γ the discount factor.

B. Set-Point Tracking

We first train a three-dimensional set-point tracking policy network to empirically check the simulation-to-reality performance of the DRL algorithms and to fly to a desired starting position for inclined landing. For this task, the states and actions are defined as follows:

$$\begin{aligned} s_{3d} &= \begin{bmatrix} x & y & z & v_x & v_y & v_z & \phi & \theta \end{bmatrix}^T, \\ a_{3d} &= \begin{bmatrix} \Theta_c & \phi_c & \theta_c \end{bmatrix}^T. \end{aligned} \tag{9}$$

Here, v is the velocity in the inertial frame. Note that the yaw angle is kept constant at zero degrees and can thus be omitted throughout all our experiments. For set-point tracking, we use the following reward function:

$$r_k = -e_p - 0.2e_v - 0.1e_{\phi,\theta} - 0.1\frac{a^2_{\phi,\theta}}{\max(e_p, 0.001)} \tag{10}$$

where e_p, e_v and $e_{\phi,\theta}$ are Euclidean distance errors of the position, velocity, and orientation, respectively, with respect to the goal state. The term $a^2_{\phi,\theta}$ is the sum of the squared roll and pitch actions (normalized between 0 and 1). It is scaled by the reciprocal of e_p to minimize oscillations near the goal position.

The policy network is a fully connected neural network with two hidden layers, with 64 neurons each, and the tanh activation function everywhere except for the output layer which has a linear activation function. The final output is subsequently clipped between -1 and 1. We use the PPO algorithm [13] to train the set-point tracking network. Other state-of-the-art DRL algorithms like twin delayed deep deterministic policy gradient (TD3) [26] and soft actor critic (SAC) [27] converged successfully as well, but PPO was superior in terms of training time and final policy performance.

C. Inclined Landing

To keep the state dimensions small and the DRL problem tractable, the inclined landing is trained in the xz-plane, representing the three-dimensional model restricted to a two-dimensional plane. The states and actions used are :

$$\begin{aligned} s_{2d} &= \begin{bmatrix} x & z & v_x & v_z & \theta \end{bmatrix}^T, \\ a_{2d} &= \begin{bmatrix} \Theta_c & \theta_c \end{bmatrix}^T. \end{aligned} \tag{11}$$

For the sake of brevity, in the sequel, we refer to s_{2d} and a_{2d} by s and a, respectively. Because the quadrotor is under-actuated, an initial swinging motion away from the landing location is required for some initial conditions. This characteristic is incompatible with the bias of a Euclidean distance based reward like the one in (10) generates. The reward function used for the inclined landing is therefore a sparse reward defined as follows:

$$r_k = \begin{cases} 0 & \text{if} \quad s_k \in S_g \\ -\beta & \text{if} \quad s_k \in S_o \\ -2 & \text{if} \quad s_k \in S_b \\ -1 & \text{otherwise.} \end{cases} \tag{12}$$

Here $S_g = \{s \mid |s_i - s_{g,i}| < \delta_{g,i}, \forall i\}$ is the set of goal states, defined as a hyperbox around the landing attitude. The goal threshold vector δ_g defines the desired landing tolerance around the goal state $s_{g,i}$ and is set by the user. The landing platform itself is an obstacle associated with a set of obstacle states S_o and a penalty β, and S_b represents the set of states close to the state space boundaries.

The use of a sparse reward requires extensive exploration to receive a non-negative reward and often leads to prolonged or unsuccessful training. We introduce the following curriculum learning [14] procedure to speed up the training and achieve convergence:

- The training starts without a landing platform and with a horizontal goal state. Only once the quadrotor reliably reaches the horizontal goal, we begin slightly tilting the goal position after each episode. Finally, the landing platform is introduced into the environment, see Fig. 2.
- We initialize simulations near the goal state and with each episode expand the set of initial positions. This eliminates the need for exploration by letting the non-negative rewards propagate throughout the value network at the beginning of training.
- We start with a large goal hyperbox S_g and as the training progresses, the hyperbox is gradually reduced to its desired size.

This learning curriculum requires an on-policy learning algorithm, such as PPO. Off-policy replay buffers would inevitably contain samples representing goals that are no longer relevant. In our experience, off-policy algorithms TD3 and SAC cannot keep up with the curriculum. The policy network architecture is similar to the one used for set-point tracking. The input and output layers for inclined landing are the state and action vectors in (11).

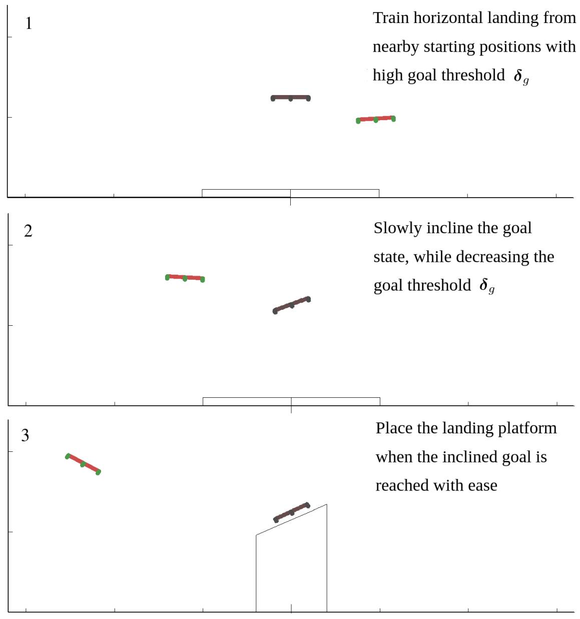

1
Train horizontal landing from nearby starting positions with high goal threshold δ_g
2
Slowly incline the goal state, while decreasing the goal threshold δ_g
3
Place the landing platform when the inclined goal is reached with ease

Fig. 2. The progress of the training curriculum for inclined landing. The red quadrotor represents the agent and the black quadrotor represents the goal state.

D. Simulation to Reality Transfer

To deploy the policies on the Crazyflie 2.1 Nano-UAV, the trained Pytorch network is converted to work within a Robot Operating System (ROS) node, which maps the state to the control input vector in (1). The positions and velocities come from a Kalman filter node, which appends the coordinates from the Optitrack motion capture system with the estimated inertial frame velocities. The quadrotor's orientation is taken from the Crazyflie's default onboard estimator. An overview of this process is given in Fig. 3.

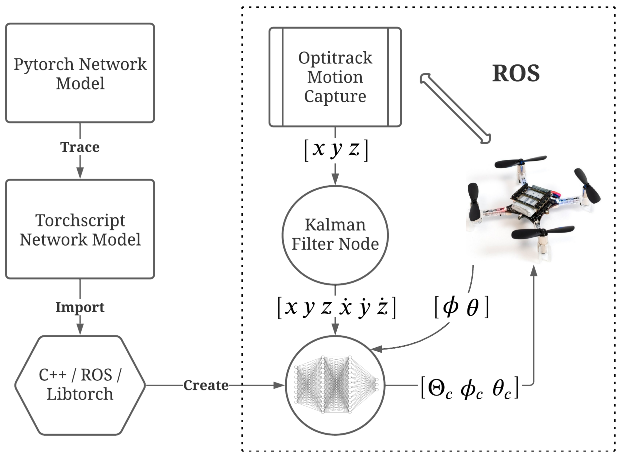


Fig. 3. Schematic overview of the Pytorch model deployment.

V. Experiments

All simulations, DRL training, and lab experiments are done on an HP Zbook Studio G4 laptop, with the default Nvidia Quadro M1200 GPU and an Intel Core i7-7700HQ CPU. The additional hardware used is the Optitrack motion-capture system, the standard Crazyflie 2.1 with a small marker holder, and a Crazyradio PA for communication with the Crazyflie.

A. Experimental Setup

The simulations and DRL training are implemented in Python, using the Pytorch package. The quadrotor dynamics are simulated using the EOM of Section III, integrated by the RK4 method. To increase the integration speed, we compile the EOM functions using Numba's Just-in-time (jit) package. By keeping the simulations and rendering in our own Gym-architecture [12] environment, we can use well-tuned implementations of existing model-free algorithms by [28]. Our compact simulator allows for computationally cheap rendering, easy curriculum adjustments and fast simulation. The simulation bounds are equal to the dimensions of the actual flying arena with $\begin{bmatrix} x_{min} & x_{max} & y_{min} & y_{max} & z_{min} & z_{max} \end{bmatrix} = \begin{bmatrix} -3.4 & 3.4 & -1.4 & 1.4 & 0 & 2.4 \end{bmatrix}$ m. The actions represent the multiplication of the clipped policy network outputs ($a \in [-1, 1]$) by the control bounds in (2), with an exception for the PWM network output which is multiplied by 16500 and added to the estimated hover PWM of 42000. All simulations use the control sampling frequency of 50 Hz, with episode length of 300 time steps, i.e., 6 seconds.

1) Set-Point Simulations: The goal state is taken as $s_g = \begin{bmatrix}0 & 0 & 1.2 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T$, which represents the center of our physical flying arena. After each episode, the initial position is set randomly anywhere within the simulation bounds, with a small margin around the edges. The PPO implementation of [28] is used, with the discount factor γ reduced to 0.97 to account for more short-term control behaviour. We train for a total of 10^6 time steps representing 27 minutes of training time. For navigation with the resulting policy, a coordinate change suffices to fly the quadrotor to an arbitrary position.

2) Inclined Landing Simulations: For inclined landing, we operate the quadrotor in the xz-plane. The DRL algorithm used is the PPO implementation of [28], where minor changes have been made to activate the rendering every 50 training iterations and to gradually start increasing γ from 0.97 to 0.99 after 300 training iterations. An episode ends at 300 time steps or when the state x is within the goal hyperbox S_g. The landing platform is modeled as a polygon and appears after $8 \cdot 10^5$ time steps. The goal threshold vector is defined as $\delta_g = \begin{bmatrix}\delta_x & \delta_z & \delta_{v_x} & \delta_{v_z} & \delta_\theta\end{bmatrix} = \begin{bmatrix}d & d & \min(10d, 1.5) & \min(10d, 1.5) & 0.25d\end{bmatrix}$, where d starts at 0.25 in the beginning of training and gradually decreases to 0.10 after 2500 episodes with 0.15/5000 per episode. The box of possible starting positions around the goal state expands with every episode by 1/6000 m in the x-direction and by 1/8000 m in the y-direction. Additionally, the goal state stays horizontal for the first $4 \cdot 10^5$ time steps and then gradually tilts towards its final inclination of $-\pi/7$ at the rate of $(-\pi/7)/6000$ radians per episode. The final goal state is set at $s_g = \begin{bmatrix}0 & 1.25 & 0 & 0 & -\pi/7\end{bmatrix}^T$. The obstacle reward constant β is taken as -7, which was found empirically to be the right trade-off between the goal of landing on top of the platform and the necessity to avoid the landing platform base. Training is stopped anywhere between $1.2 \cdot 10^6$ and $3 \cdot 10^6$ time steps (30 to 80 minutes), when the rendering shows that the policy executes the inclined landing reliably. Note that rather than monitoring the loss function, the aforementioned parameters and curricula have been empirically tuned by frequently rendering. The trend in the loss function value is quite meaningless, given the curricula and the discount factor adaptation.

3) Validation on a real quadrotor: The quadrotor used is the Crazyflie 2.1 Nano-UAV with its original firmware. A Crazyflie-specific package [21] allows us to publish the control values in (1) directly to the ROS server, which are then transmitted with low latency to the Crazyflie over the Crazyradio PA. The trained policy network is evaluated at 80 Hz, even though it was trained at 50 Hz. This is possible, as the policy is a function of the physical state only, and can therefore be evaluated at any frequency. A single policy evaluation takes approximately 2.5 ms. The coordinates from the Optitrack motion-capture system come in at 120 Hz and are combined with the inertial frame velocities by the Kalman filter node [29]. The orientation is received from the Crazyflie's onboard estimator through the Crazyradio at around 80 Hz. The position, velocity, and orientation estimates form the policy input. The experiment itself starts with the drone flying to a position of choice, using the three-dimensional set-point tracking network. Once it is positioned, the networks are switched and the drone commences the inclined landing. The initial state can be at an arbitrary location in the top half of the flying arena. The landing trial ends when the quadrotor's state s_k reaches the set of goal states S_g.

B. Experiment Results

Early experiments showed a vertical offset between the simulated and real trajectories, caused by a slight inaccuracy of the motor thrust equation (6). We compensated for this offset by increasing the hover PWM to 48000 during the flying arena testing. The resulting behaviour was very similar to the simulations, as can be seen in Fig. 4.[5] These trajectories originate from the same policy network, starting from arbitrary initial positions and ending when $s_k \in S_g$ with $\delta_g = \begin{bmatrix}0.10 & 0.10 & 1.5 & 1.5 & 0.025\end{bmatrix}$.

To further evaluate the performance, we measured the landing success rate when starting the flight from three different initial positions. Each position was evaluated 10 times, resulting in the success percentages reported in Table I. Even if an experiment did not succeed, the agent would not crash but autonomously fly back and forth, sometimes succeeding in its second or third attempt. However, these further attempts were not counted as a successful landing.

TABLE I
COMPARISON OF REAL-WORLD AND SIMULATION EXPERIMENTS

setting	success from initial (x, z)			**total**
	$(0, 2)$	$(-1.5, 1.6)$	$(1.5, 1.8)$	
Real-World	90%	70%	100%	**86.7%**
Simulation	90%	90%	100%	**93.3%**

No simulation-to-reality transfer techniques, such as domain randomization, have been employed, as they did not seem to improve the final performance, while they did complicate the agent's training. Additionally, we found that using the Crazyflie's onboard orientation estimates rather than the Optitrack orientation estimates resulted in a substantial improvement in the consistency of performance.

The results show that inclined landing controllers can be designed by means of DRL. These controllers can transfer adequately to reality without the need for dynamics randomization, or sensor noise. Furthermore, the same policy can be executed from a wide variety of initial states. The performance could further improve by additional system

[5]Because of a faulty Optitrack measurement, the bottom figure shows a single misplaced red quad with a corresponding short drop in the PWM output.

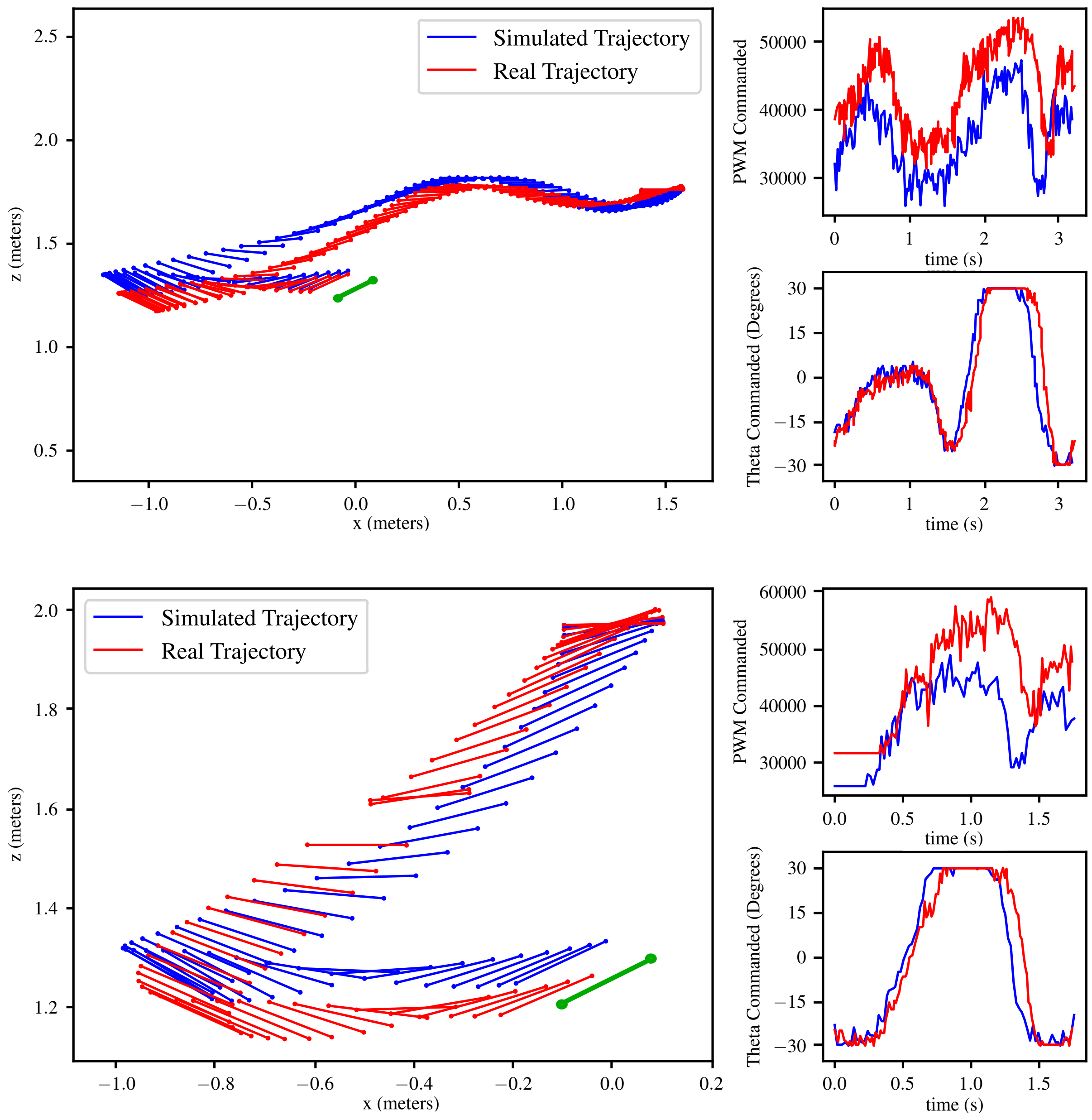


Fig. 4. Landing trajectories starting from above right (top) and from directly above (bottom) of the inclined goal position (green) at (0, 1.25). The trajectory ends when the quadrotor is within the set of goal states S_g.

identification of the total motor thrust. The slight mismatch observed might have been caused by making too strong assumptions about the scaling of the single motor model in (5) to the full motor model in (6). However, a perfect thrust model will never exist, due to the motor wear and tear and the strong relation of the drone's battery level to its thrust output, which was also reported in [1].

VI. DISCUSSION

A. *Onboard Controller Dynamics*

Using a model that incorporates the inner-loop system dynamics of the Crazyflie's onboard controllers allowed us to fly the quadrotor by using intermediate-level control commands. Although this requires a quadrotor-specific parameter identification, the procedure was straightforward (as also reported in [25]), took a negligible amount of time, and can be applied to any onboard-controlled quadrotor. Furthermore, quadrotor end-users usually prefer to keep the onboard controllers in place, as they provide basic functionality and safety features. This has therefore been the main reason we conducted the research in this setting. A drawback of this approach compared to training DRL on individual motor thrusts [1], [4] has been the limit of 30° of the attitude controller, restricting us to maximum landing angles of around $25.7°$. However, the benefit of using the closed-loop model is that the policy does not need to stabilize the quadrotor in the first place, and can focus on using the quadrotor's stable dynamics to learn the behaviour needed for the inclined landing.

B. DRL Algorithms and Curricula

For set-point tracking, PPO was far superior to TD3 or SAC in terms of performance and training time. For inclined landing, we have tested these off-policy algorithms with reduced replay buffers to cope with the changing goal state (between $1 \cdot 10^4$ and $3 \cdot 10^4$ samples). The resulting policies were very poor, shaky and not capable of yielding a smooth landing behaviour in the last curriculum phase.

In the beginning of training, having a larger goal threshold δ_g was important for convergence, even when starting at and around the goal state. Having a too large goal velocity threshold would however cause strong oscillations during training, hence the minimum term in the training value for δ_{v_x} and δ_{v_z}. The convergence during the subsequent inclined landing phase was mostly dependent on the performance during the horizontal phase. As long as the quadrotor could adequately reach the horizontal hover position, it would be able to transition into larger angles by means of small increments. Interestingly, this is where the quadrotor learns the swinging behaviour on its own, making use of its inherent dynamics. The success of the final phase depends on the obstacle penalty β, where a small value will make the agent exploit the platform for braking, and a large value will diminish the incentive to reach the goal state. The final curriculum works for landing angles of $25.7°$ with an attitude controller limitation of $30°$, but simulations using a hypothetical attitude controller limitation of $55°$ have showed that angles of $50°$ can also be reached within the same time-span and using the same training procedure.

C. Landing in the vertical xz-plane

Because the landing controller is planar, it will not compensate drift in the y-axis. Since our experiments were conducted indoors, this problem was negligible. However, when conducting experiments under disturbances or with poorly calibrated quadrotors, one could add a basic PID controller regulating the alignment with the y-axis.

We believe that the planar controller is still very applicable in real settings. For inclined landings, as we envisage them, the landing platform will almost always be approachable in a plane (exceptions would be strong disturbances like side wind or complicated obstacles in the approach trajectory). This means that a quadrotor can fly toward the platform, change its yaw axis accordingly, and perform the inclined landing using the planar controller.

VII. CONCLUSION AND FUTURE WORK

We have presented a model-free DRL technique to facilitate autonomous quadrotor landing on an inclined surface. We trained a control agent with PPO, using sparse rewards and a learning curriculum. This allows the agent to gradually progress toward a more difficult task, i.e., larger inclination angles of the landing platform. Moreover, we have shown that the trained policies transfer well to reality, without employing any simulation-to-reality transfer techniques.

A limitation of this work is the fact that we restricted the landing trajectory to the xz-plane, which may cause some drift in the y-direction. A three-dimensional landing policy could increase precision, albeit at the cost of longer and more complex training. An extension to such a setting is the topic of our future work. Preliminary simulation results show that training a three-dimensional policy is feasible and converges within a similar time span as the two-dimensional policy. Additional future work could focus on larger inclination angles of the landing platform, as long as the onboard attitude controller would allow them. In this way, one could try to extend the findings in this paper to a perching behavior similar to [20]. The goal inclination can also be added to the quadrotor's state, which would enable landing on unknown platforms, using, for instance, a laser measurement system [18] or an onboard camera system [19]. Another interesting path for future work is to train a single inclined landing policy that is applicable on multiple quadrotors with comparable onboard control architectures, similar to the work in [4]. Finally, a form of the platform contact dynamics in [30] could be implemented in our simulator for a more robust landing and to aid the design of an end-to-end controller, which would eliminate the need for an external stopping signal.

ACKNOWLEDGMENT

We thank our colleague Hai Zhu for his guidance and advice in the initial stage of this project. Robert Babuška was supported by the European Union's H2020 project Open Deep Learning Toolkit for Robotics (OpenDR) under grant agreement No. 871449.

REFERENCES

[1] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 06 2017.

[2] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 2, Feb. 2019.

[3] N. O. Lambert, D. S. Drew, J. Yaconelli, R. Calandra, S. Levine, and K. S. J. Pister, "Low level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, pp. 4224–4230, 2019.

[4] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," *arXiv preprint arXiv:1903.04628*, 2019.

[5] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9784–9790.

[6] Y. Bicer, M. Moghadam, C. Sahin, B. Eroglu, and N. K. Üre, "Vision-based uav guidance for autonomous landing with deep neural networks," in *AIAA Scitech 2019 Forum*, 2019, p. 0140.

[7] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward end-to-end control for uav autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 115–123.

[8] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, "Autonomous vehicular landings on the deck of an unmanned surface vehicle using deep reinforcement learning," *Robotica*, vol. 37, no. 11, p. 1867–1882, 2019.

[9] R. Polvara, M. Patacchiola, M. Hanheide, and G. Neumann, "Sim-to-real quadrotor landing via sequential deep q-networks and domain randomization," *Robotics*, vol. 9, no. 1, 2020.

[10] A. Rodríguez Ramos, C. Sampedro Pérez, H. Bavle, P. de la Puente, and P. Campoy, "A deep reinforcement learning strategy for uav autonomous landing on a moving platform," *Journal of Intelligent & Robotic Systems*, vol. 93, 02 2019.

[11] M. B. Vankadari, K. Das, C. Shinde, and S. Kumar, "A reinforcement learning approach for autonomous control and landing of a quadrotor," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 676–683.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 41–48.

[15] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[16] D. Ghosh, A. Gupta, A. Reddy, C. M. Devin, B. Eysenbach, and S. Levine, "Learning to reach goals via iterated supervised learning," in *International Conference on Learning Representations*, 2021.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR*, 2016.

[18] J. Dougherty, D. Lee, and T. Lee, "Laser-based guidance of a quadrotor UAV for precise landing on an inclined surface," *2014 American Control Conference*, pp. 1210–1215, 2014.

[19] P. Vlantis, P. Marantos, C. P. Bechlioulis, and K. J. Kyriakopoulos, "Quadrotor landing on an inclined platform of a moving ground vehicle," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2202–2207.

[20] J. Thomas, M. Pope, G. Loianno, E. Hawkes, M. Estrada, H. Jiang, M. Cutkosky, and V. Kumar, "Aggressive flight for perching on inclined surfaces," *Journal of Mechanisms and Robotics*, vol. 8, 2015.

[21] W. Hönig and N. Ayanian, *Flying Multiple UAVs Using ROS*. Springer International Publishing, 2017, pp. 83–118.

[22] Y. Chen and N. O. Pérez-Arancibia, "Nonlinear adaptive control of quadrotor multi-flipping maneuvers in the presence of time-varying torque latency," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[23] J. Förster and R. D'Andrea, "System identification of the crazyflie 2.0 nano quadrocopter," Zürich, Tech. Rep., 2015.

[24] M. W. Mueller, M. Hamer, and R. D'Andrea, "Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1730–1736.

[25] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 236–243.

[26] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *ICML*, 2018, pp. 1582–1591.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 10–15 Jul 2018, pp. 1861–1870.

[28] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines 3," https://github.com/DLR-RM/stable-baselines3, 2019.

[29] H. Zhu and J. Alonso-Mora, "Chance-Constrained Collision Avoidance for MAVs in Dynamic Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 776–783, 2019.

[30] J. Bass and A. L. Desbiens, "Improving multirotor landing performance on inclined surfaces using reverse thrust," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5850–5857, 2020.

C

IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 6, NO. 4, OCTOBER 2021 6289

Learning Kinematic Feasibility for Mobile Manipulation Through Deep Reinforcement Learning

Daniel Honerkamp, Tim Welschehold, and Abhinav Valada

Abstract—**Mobile manipulation tasks remain one of the critical challenges for the widespread adoption of autonomous robots in both service and industrial scenarios. While planning approaches are good at generating feasible whole-body robot trajectories, they struggle with dynamic environments as well as the incorporation of constraints given by the task and the environment. On the other hand, dynamic motion models in the action space struggle with generating kinematically feasible trajectories for mobile manipulation actions. We propose a deep reinforcement learning approach to learn feasible dynamic motions for a mobile base while the end-effector follows a trajectory in task space generated by an arbitrary system to fulfill the task at hand. This modular formulation has several benefits: it enables us to readily transform a broad range of end-effector motions into mobile applications, it allows us to use the kinematic feasibility of the end-effector trajectory as a dense reward signal and its modular formulation allows it to generalise to unseen end-effector motions at test time. We demonstrate the capabilities of our approach on multiple mobile robot platforms with different kinematic abilities and different types of wheeled platforms in extensive simulated as well as real-world experiments.**

Index Terms—**Mobile manipulation, reinforcement learning.**

I. Introduction

MOBILE manipulation is a key research area on the journey to both autonomous household assistants as well as flexible automation processes and warehouse logistics. Although impressive results have been achieved over the last years [1]–[4], there remain multiple unsolved research problems. One of the major ones being that most current approaches separate navigation and manipulation due to the difficulties in planning the joint movement of the robot base and its end-effector (EE). This restricts the range of tasks that can be solved and constrains the overall efficiency that can be achieved. Typically, the tasks that a robot is expected to perform are linked to conditions in the task space, such as poses at which handled objects can be grasped, orientation that objects should maintain or entire trajectories that must be followed. While there are techniques to position a manipulator to fulfill various task constraints with respect to the kinematics of the robot, based on inverse reachability maps (IRM) [5], performing such tasks while moving the base still remains an unsolved problem.

Classical planning approaches circumvent kinematic issues implicitly by exploring paths in the configuration space of the robot [6]. However, this creates a number of new problems. First, the constraints must be transferred from the task space to the robot specific configuration space, requiring expert knowledge on the task, the robot and the environment. Furthermore, the execution of pre-planned configuration space movements in dynamic environments is challenging as minor errors in the execution of poses in the configuration space can lead to large deviations in the task space. Moreover, adjustments to the movement might be necessary due to changes in the dynamic scene which requires complete re-planning in the configuration space.

In this paper, we present a method to generate kinematically feasible movement for the base of a mobile robot while its end-effector executes motions generated by an arbitrary system in the task space to perform a certain action. This decomposes the task into generation of trajectories for the end-effector, which is typically defined by the task constraints, and the robot base, which should handle kinematic constraints and collision avoidance. This separation is beneficial for many robotic applications. First, it results in high modularity where action models for the end-effector can easily be shared among robots with different kinematic properties. Therefore, there is no need to adapt the behavior of the end-effector for different robots as the kinematic constraints are handled entirely by the base motion control. Second, if the learned base policy is able to generalise to arbitrary end-effector motions, the same base policy can be used on new unseen tasks without expensive retraining. Lastly, mobile manipulation tasks are complex, long horizon problems that can be difficult to learn with sparse rewards only. We show that we can directly leverage the kinematic feasibility as a dense reward signal across platforms, alleviating the need for extensive reward shaping.

In prior work [7], we addressed the kinematic feasibility of a joint base and end-effector motion by treating the inverse reachability constraint as an obstacle avoidance problem. While the approach achieves good results in mobile manipulation actions on a PR2 robot, it requires substantial robot specific design and the approximations made to model the inverse reachable space restricts the movement of the robot further than necessary. Instead of explicitly modeling the kinematic abilities, we propose to directly learn a feasible motion of the robot base respecting

Manuscript received January 14, 2021; accepted June 14, 2021. Date of publication June 25, 2021; date of current version July 15, 2021. This letter was recommended for publication by Associate Editor L. Natale and Editor M. Vincze upon evaluation of the reviewers' comments. This work was supported by the European Union's Horizon 2020 Research and Innovation Program under Grant Agreement No. 871449-OpenDR and a Research Grant from Eva Mayr-Stihl Stiftung. *(Corresponding author: Daniel Honerkamp.)*

The authors are with the Department of Computer Science, University of Freiburg, Germany (e-mail: daniel.honerkamp@gmail.com; twelsche@cs.uni-freiburg.de; valada@informatik.uni-freiburg.de).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2021.3092685, provided by the authors.
Digital Object Identifier 10.1109/LRA.2021.3092685

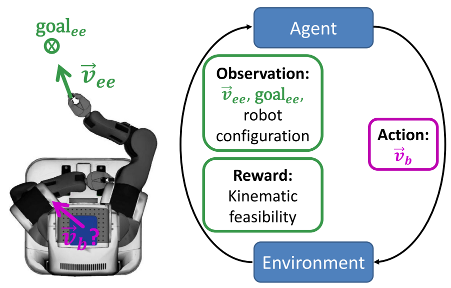

goal$_{ee}$
$\vec{v}_{ee}$
$\vec{v}_b$
Agent
Observation: $\vec{v}_{ee}$, goal$_{ee}$, robot configuration
Reward: Kinematic feasibility
Action: $\vec{v}_b$
Environment

Fig. 1. Given the robot configuration, a velocity for the end-effector $\vec{v}_{ee}$ and the desired goal for the end-effector, the robot learns a corresponding base velocity $\vec{v}_b$ in a reinforcement learning setting to maintain kinematic feasibility throughout the motion execution.

a given motion of the end-effector and show that the learned policy strongly outperforms these approximations. We illustrate our envisioned system in Fig. 1.

In summary, we make the following main contributions:

1) We formulate the fulfillment of kinematic feasibility constraints in mobile manipulation tasks as a reinforcement learning problem.
2) We design multiple environments for different robots with considerably different kinematic abilities and varying driving modes.
3) We present extensive simulated and real world experiments which demonstrate that our approach itself generalises across diverse robotic platforms while the platform specific trained models generalise to seen and unseen end-effector motions.
4) We make the source code, models and videos publicly available at http://rl.uni-freiburg.de/research/kinematic-feasibility-rl.

II. Related Work

In general, there are two distinct methods to ensure kinematic feasibility in mobile manipulation tasks. On one hand, planning frameworks can be used to plan trajectories for the robot in joint space and thereby only explore kinematically feasible paths [6], [8]. On the other hand, inverse reachability maps [9] can be used to seek good positioning for the robot base given the task constraints [5]. While combinations of the two methods exist [10], it remains a hard problem to integrate kinematic feasibility constraints in task space mobile motion planning.

In this context, Welschehold *et al.* [7] propose a geometric description of the inverse reachability and address the kinematic feasibility of an arbitrary gripper trajectory as an obstacle avoidance problem. They first approximate feasible base poses relative to the end-effector resulting in a bounded region for the base. They then analytically modulate the base velocity to stay within feasible regions and orientations. Although their approach performs well on a real-world PR2 robot, the approximations require expert knowledge and do not easily generalise to different platforms. On a high level, we use a conceptually similar setup of given EE-motions and base control. We do not impose any geometric constraints on the allowed base poses by using reinforcement learning (RL) to directly learn the base velocities. We also show that our approach generalises to different robots without the need for robot-specific expert knowledge, i.e. can directly be used to train agents for each platform. We directly compare with [7] in our experiments.

While RL has shown substantial promise in manipulation tasks, it has only recently been incorporated into mobile manipulation tasks. RelMoGen proposes to learn high-level subgoals through RL to simplify the exploration problem [4]. It focuses on tasks in which the exact gripper trajectories are not relevant and learns to either move the gripper or the base. In contrast, we are explicitly interested in task specific end-effector trajectories and use RL to to ensure kinematic feasibility of conjoint end-effector and base motions.

Kindle *et al.* [11] use RL to learn both base and end-effector movements end-to-end but they restrict the arm to lie on a plane parallel to the ground. Using a handcrafted reward with numerous hyperparameters, they demonstrate navigation in a hallway. Similarly, Wang *et al.* [12] solve a mobile picking task by learning to jointly control both the base of the robot and its end-effector. While these works focus on learning a policy for one specific task, we address a more general problem of maintaining kinematic feasibility in arbitrary mobile manipulation actions. By introducing an arbitrary end-effector motion planner, we decouple the RL agent's behaviour from the exact task that the end-effector performs. This enables us to use kinematic feasibility as the sole reward signal and thereby generalises to different tasks with different motion constraints. Recently the use of RL has also been explored to calculate forward and inverse kinematics of complex many-joint robot arms [13], [14]. While such approaches are interesting for robot system with a large number of degrees of freedom (up to 40-dof [14]), in our applications (7-dof in robot arms) the inverse kinematics can still be solved numerically.

Goal conditional RL [15], [16] takes both the current state and a goal as input to predict the actions to arrive at this goal. Hierarchical methods [17]–[19] commonly reduce the complexity of long-horizon tasks by splitting it into a subgoal proposal and goal-conditional policy. Li *et al.* [20] adapt hierarchical RL to mobile manipulation by incorporating an additional high-level action that restricts the low-level action space to either the base, arm or both, and demonstrate its ability to reach goals behind closed doors. While it is more-data efficient than previous hierarchical approaches, it still has to deal with the added complexity of the non-stationarity between different levels, and the additional parameters and rewards to learn both levels. In contrast, we reduce the complexity of the task by assuming a given planner for a subset of the agent (the gripper) and learn to control the remaining degrees of freedom (the base) to achieve the generated end-effector motions. This shifts the burden from the end-effector planner to the base policy and allows us to use simple and general methods to generate the end-effector motions.

III. Learning Kinematically Feasible Robot Base Motions

Mobile manipulation tasks require complex trajectories in the conjoint space of arm and base over long horizons. We decompose the problem into a given, arbitrary motion for the

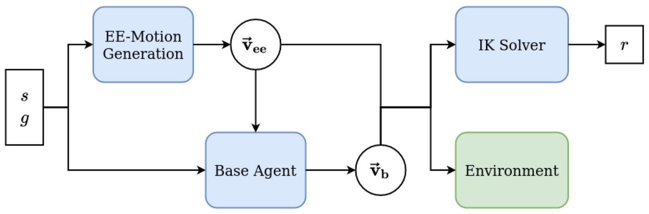


Fig. 2. We decompose mobile manipulation tasks into two components: an end-effector (EE) motion generation and a conditional RL agent that controls the base velocities. This enables us to produce a dense reward solely based on the kinematic feasibility, computed by a standard inverse kinematics solver.

end-effector and a learned base policy. This allows us to readily transform end-effector motions into mobile applications and to strongly reduce the burden on the end-effector motion generation which can now be reduced to a fairly simple system. We then formulate this as a goal-conditional RL problem and show that we can leverage kinematic feasibility as a simple, dense reward signal instead of relying on either sparse or extensively shaped rewards. Our proposed system consists of three main components: a motion generator for the end-effector, a learned RL policy for the base and a standard inverse kinematics (IK) solver for the manipulator arm that provides us with the rewards. An overview of the system is shown in Fig. 2.

A. *End-Effector Trajectory Generation*

To generate end-effector motions, we assume access to an arbitrary motion generator that takes as input the current end-effector pose and a goal state g in a fixed map frame, specified by an end-effector pose in cartesian space. At every time step, this generator then outputs the next velocity command $\mathbf{v}_{ee}$ for the end-effector. During training we pass the last *desired* instead of the current EE pose to the generator as to prevent the RL agent from influencing the shape of the overall EE-trajectory. At test time the EE- motion generator can be replaced by an arbitrary system.

B. *Learning Robot Base Trajectories*

Given an end-effector motion dynamic, our goal is to ensure that the resulting EE-poses remain kinematically feasible at every time step. We can formulate this as a goal-conditioned reinforcement learning problem. We define a finite-horizon Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$ with state and action spaces $\mathcal{S}$ and $\mathcal{A}$, transition dynamics $P(s_{t+1}|s_t, a_t)$, a reward function r and a discount factor γ. The objective is to find the policy $\pi(a|s)$ that maximises the expected return $\mathbb{E}_\pi[\sum_{t=1}^T \gamma^t r(s_t, a_t)]$. We can extend this to a goal-conditional formulation [15] to learn a policy $\pi(a|s, g)$ that maximises the expected return $r(s_t, a_t, g)$ under a goal distribution $g \sim \mathcal{G}$ as $\mathbb{E}_\pi[\sum_{t=1}^T \gamma^t r(s_t, a_t, g)]$.

At every step, the agent observes the current state s_t consisting of the current arm joint configuration as well as the current end-effector pose, the next generated gripper velocities and the end-effector goal g, all in the robot's base frame. Rotations and changes in rotation are represented by normalised quaternions, resulting in a state space of dimension $21 + n_{joints}$ where n_{joints} is the number of articulated joints in the robot's arm. It then learns a policy $\pi(s_t, g)$ for the next base velocity commands $\mathbf{v_b}$. We also experimented with learning the parameters of a geometric modulation as introduced in [7] or velocities relative to the end-effector velocity but found directly learning the base commands to be more robust.

C. *Kinematic Feasibility*

By separating EE and base motions, we can now directly leverage the kinematical feasibility of the EE-motions to train the base policy. We convert this into a straightforward reward function that simply penalises whenever an EE-motion is not feasible. This provides us with a dense reward signal without having to rely on ground truth distances or extensive reward shaping. Instead it naturally arises from framing mobile manipulation as a modular problem. We also add a regularisation term to keep the actions small whenever possible to avoid unnecessary or extensive base movements. The overall reward function can be expressed as

$$r(s, a, g) = -\mathbb{1}_{!kin} - \lambda||\mathbf{a}||^2, \tag{1}$$

where $\mathbb{1}_{!kin}$ is an indicator function evaluating to one when the next gripper pose is not kinematically feasible and λ is a hyperparameter weighting the squared norm of the actions. To evaluate the kinematic feasibility, we first compute the next desired end-effector pose from the current pose and velocities. We then use standard kinematic solvers to evaluate whether this new pose is feasible.

We optimise this objective with recent model-free RL algorithms that have shown to be robust to noise and overestimation, namely TD3 [21] and SAC [22]. An episode ends when either the end-effector pose is within 2.5 cm of the goal or more than 19 kinematic failures have occurred (99 during evaluation). The result is a reward function that can be evaluated without any adaptation across a wide variety of platforms. It furthermore allows us to use the same learned behaviours across a wide range of tasks as the RL objective is agnostic to the nature of the task itself. In our experiments, we show that EE-motions for many common robotic tasks can be easily derived from existing motion systems or constructed with very simple methods – as we can now abstract from the feasibility of the motion.

D. *Training Task*

The motivation for modularising end-effector and base control is to learn a base policy that enables a large number of task-specific end-effector trajectories. In many cases we will not know all tasks at training time. To be able to generalise to diverse end-effector motions the agent should ideally observe a wide range of different relative EE-poses, motions and goals. To do so, we train the agent on a random goal reaching task. We first initialize the robot in a random joint configuration and then uniformly sample end-effector goals within a distance of one to five meters around the robot base and from the full range of reachable heights.

To generate end-effector motions, we use a *linear dynamic system* where the end-effector velocity for each step is generated as the difference between the current pose and the sampled goal, constrained by a minimum and a maximum velocity. For the orientation part we use spherical linear interpolation (slerp). By training with a very simple EE- motion generator that does not take into account the current joint configuration, we shift the burden of generating feasible kinematic movements to the base policy. During training we add a small Gaussian noise with a standard deviation of 1.5 cm/s to base velocities to increase robustness to imperfect motion executions in the real world.

IV. Experimental Evaluation

We evaluate our approach on multiple mobile robot platforms in a series of analytical, simulated and real-world experiments to address the following questions:

- Does our approach generalise across robotic platforms with different kinematic abilities?
- Do the learned policies generalise to task-specific gripper motions from both seen and unseen EE-motion generators?
- Do the analytically learned policies transfer to execution in simulation and the real world?

A. Experimental Setup

1) Robot Platforms: We train agents for three different robotic platforms differing considerably in their kinematic structure and base motion abilities. The *PR2* robot is equipped with a 7-DOF arm mounted on an omnidirectional base, giving it high mobility and kinematic flexibility. The *TIAGo* robot is also equipped with a 7-DOF arm and we additionally use the height adjustment of its torso. For the base motion it uses a differential drive restricting its mobility compared to the PR2. The *Toyota HSR* robot also has an omnidirectional base but the arm is limited to 5-DOF including the height adjustable torso. Given the low flexibility of the HSR arm, we consider distances of up to 10 cm and angles of up to 12° to the desired EE-poses as kinematically feasible. This leeway does not apply to the final goal. To minimize the use of this leeway, we additionally penalize the sum of the squared distance and angular distance to the desired EE-pose, scaled into the range of $[-1, 0]$ (i.e. smaller or equal to the penalty for kinematically infeasible poses). The action space for these platforms is continuous, consisting of either one (diff-drive) or two (omni) directional velocities $\mathbf{v}_{\mathbf{b},\{\mathbf{x},\mathbf{y}\}}$ and an angular velocity $\mathbf{v}_{\mathbf{b},\theta}$. Table I shows the constraints we set across the different platforms in the analytical environment.

2) Tasks: We construct five tasks: A *general goal reaching (ggr)* task in which the goals are selected randomly as in the training phase. As the kinematics become very restrictive towards the edges of the height range, we also analyse the results for initial configurations and goals that are restricted to more common heights, which we refer to as *ggr restr*. Table I shows the values that we specify for the different robots. A *pick&place* task in which the robot has to grasp an object randomly located on the edge of a table, move it to a different goal table randomly located on another wall in the room and place it down. We use the linear system to generate EE-motions by sequentially

TABLE I
Velocity Constraints for the Different Robot Platforms and Components in the Analytical Environment. Constraints in the Physics Simulator Are Defined by the Respective Default Trajectory Controllers. Height Constraints Refer to the *Ggr* and *Ggr Restr* Tasks and Are Defined for the Wrist Link of the Robot

Parameter	EE-Motion	PR2	TIAGo	HSR
Max. velocity (m/s)	0.1	0.2	0.2	0.2
Max. rotation (rad/s)	0.1	1.0	0.4	1.5
Goal height (m)	-	[0.2, 1.4]	[0.2, 1.5]	[0.2, 1.4]
Restr. height (m)	-	[0.4, 1.0]	[0.4, 1.1]	[0.4, 1.1]

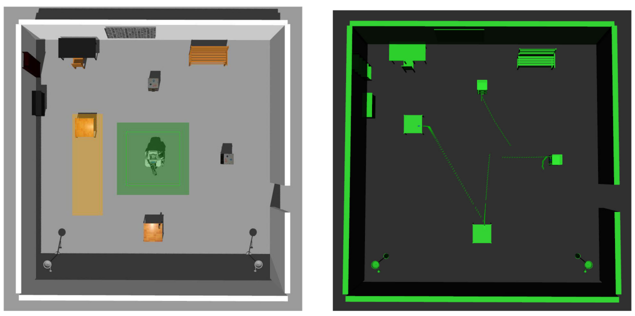


Fig. 3. Left: The robot starts in a random location and rotation in the center of the room (green). The task objects consisting of tables and shelves equipped with doors and drawers are located around the robot. The random location of the drop-off table is marked in orange. Right: Examples of the different generated EE-motions for each task. Markers indicate the desired EE-pose at every 10^{th} step.

combining four goals relative to the object: slightly in front of the object, at the object, in front of and at the goal location. To test whether our approach generalises to different EE-motions we then construct two more tasks from an unseen *imitation learning system* developed in [23]. These motions are learned from a human teacher and encoded in a dynamic system following a demonstrated hand trajectory to manipulate a certain object. As a result, the motions can differ substantially from the linear system used during training. We use models to *grasp and open a cabinet door* as well as to *grasp and open a drawer*. The corresponding motion models are autonomously adapted to the given poses of the handled objects. Fig. 3 (right) shows examples of the generated EE-motions for each of the tasks.

For the virtual evaluation we locate the objects in a room around the robot as shown in Fig. 3 (left). In all tasks, the robot starts randomly within a 1.5×1.5 m square in the center of the room, rotated between $[-\pi/2, \pi/2]$ rad relative to the first end-effector goal and in a random joint configuration sampled from the full possible configuration space, including difficult and unusual poses.

3) Baselines: We construct separate baselines for the linear system and the imitation system. For the linear system the baseline replicates the same robot base motion in $xy-$direction as the end-effector. For Tiago this agent also takes into account the limitations of the differential drive. This simple strategy removes some of the main difficulties by keeping a fixed distance between EE and body, avoiding situations in which the gripper would have to pass around or through the robot body. The baseline

TABLE II
HYPERPARAMETERS SEARCHED, VALUES CHOSEN BASED ON GRID SEARCH ON THE TRAINING TASK. WE USE A PUBLIC IMPLEMENTATION OF THE TD3 AND SAC ALGORITHMS [24]. PARAMETERS NOT MENTIONED ARE LEFT AT THEIR DEFAULT VALUES, INCLUDING THE ACTOR AND CRITIC NETWORKS OF TWO FULLY-CONNECTED LAYERS SIZE (256, 256) FOR SAC AND (400, 300) FOR TD3. IK FAIL THRESH IS THE MAXIMUM NUMBER OF FAILURES BEFORE WE TERMINATE THE EPISODE AND LR IS THE LEARNING RATE

Parameter	Values	Parameter	Values
Algorithm	{SAC, TD3}	τ	{0.001, 0.005}
Batch size	{64, 256}	γ	{0.98, 0.99, 0.999}
Ik fail thresh	{1, 19, 99}	ϵ-noise	{0.25, 0.5, 0.75}
λ	{0.0, 0.01, 0.1}	Rnd steps	{0, 50'000}
Lr	{3e−4, 1e−4, 1e−5 }	Policy noise	{0.1, 0.25, 0.5}
Lr decay	{0.999}	Entropy reg	{learn, 0.1, 0.2, 0.3}
Buffer size	{100'000}		

for the imitation system follows base motions that were learned together with the end-effector, adapted to the different lengths of the robot arms. We combine these two baselines under the terms *PR2_bl, Tiago_bl* and *HSR_bl*. We also compare against the geometric modulation for the PR2 presented in [7], termed *PR2_gm*. This approach learns an approximation to the inverse reachability in closed form and modulates the base velocities to stay within allowed poses. As the approximations rely on robot specific knowledge, it cannot easily be adapted to the other platforms.

4) Training Details and Metrics: For each robotic platform, we conduct a hyperparameter search over the parameters listed in Table II. We then select the best configuration and train the agent on new seeds for roughly 2000 episodes (1000000 steps) of the random goal reaching task described in Sec. III. For each platform we train models on ten different seeds and average the results. We then evaluate the agent's ability to achieve the described tasks without any task-specific fine-tuning over 50 episodes per task. For each task, we report the share of the trajectories executed without a single kinematic failure. Note that this is a fairly strict metric and in many cases even with a few failures, the task can still be completed successfully. For this reason, we also report the share of episodes which never deviate more than 5 cm from the EE-motions (in brackets). We find that both TD3 and SAC learn to solve the tasks successfully, but SAC generally results in more robust policies and ultimately slightly better performance. Therefore we use SAC throughout the remainder of this work.

B. Analytical Evaluation

We train the model on analytically generated trajectories, i.e. we integrate the system step-by-step to generate the poses of robot end-effector and base. At each step we evaluate the kinematic feasibility of the generated poses without a physical simulation of robot controllers. We evaluate this system at a step size corresponding to a frequency of 10 Hz. Fig. 4 shows the success rates over the course of the training, averaged over ten seeds for each of the platforms. As the kinematic feasibility provides us with a dense reward, the agents reach reasonable performance already within a few hundred episodes (an average training episode lasts roughly 500 steps).

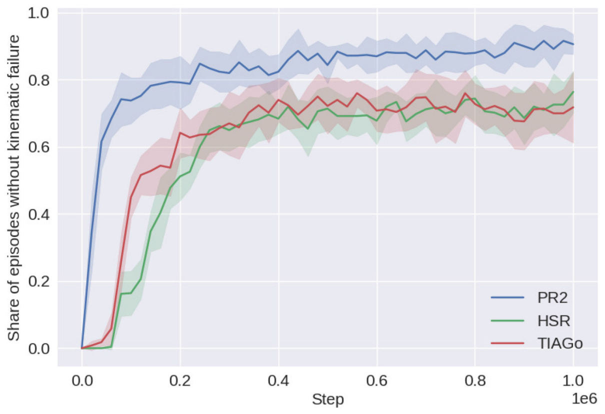


Fig. 4. Training progress measured as share of episodes with zero kinematic failures on the random goal reaching task. All agents were trained with SAC, evaluated over 50 episodes and averaged over 10 seeds. Shaded regions show the standard deviation.

TABLE III
PERFORMANCE IN THE ANALYTICAL EVALUATION AS SHARE OF SUCCESSFULLY EXECUTED EPISODES WITH ZERO KINEMATICALLY INFEASIBLE EE-POSES AND SHARE OF EPISODES THAT NEVER DEVIATE MORE THAN 5 CM FROM THE EE-MOTION (IN BRACKETS). THE BOLD ROWS REPRESENT OUR PROPOSED APPROACH. WE EVALUATE EACH TASK OVER 50 EPISODES FOR 10 RANDOM SEEDS

Agent	Linear Dynamic System			Imitation Learning	
	ggr	ggr restr	pick&place	door	drawer
PR2_bl	60.8 (65.0)	70.8 (72.8)	72.6 (76.8)	30.2 (33.6)	31.6 (35.4)
PR2_gm	64.6 (69.6)	68.6 (74.4)	74.4 (78.6)	31.6 (38.0)	28.2 (34.0)
PR2	**90.2 (91.2)**	**88.8 (90.6)**	**97.0 (97.4)**	**94.2 (95.4)**	**95.4 (95.4)**
Tiago_bl	21.6 (24.6)	23.6 (26.8)	12.0 (13.8)	9.2 (10.0)	28.6 (31.8)
Tiago	**71.6 (73.4)**	**80.2 (83.0)**	**91.4 (92.2)**	**95.3 (96.9)**	**94.9 (95.3)**
HSR_bl	5.4 (7.4)	7.6 (10.4)	0.0 (3.2)	0.0 (0.0)	0.0 (0.0)
HSR	**75.2 (69.8)**	**80.1 (72.6)**	**93.4 (90.2)**	**91.2 (87.0)**	**90.6 (88.6)**

We then evaluate the trained models across all tasks in the same environment. The results are summarised in Table III. On the PR2 the baseline that replicates the EE-velocities is able to complete between 60% and 73% of the linear motion tasks successfully. By simply "sliding" towards the goal, the manipulation task is kept relatively static and most of the difficulty is transferred to the IK solver. As a consequence, performance on more constraint platforms drops to between 0% and 24% for both the 5-DOF arm of the HSR as well as the diff-drive of TIAGo that has to "circle" towards the goal. This illustrates the need for base and arm to work together to achieve these motions. On the door and drawer tasks, the PR2 and TIAGo are able to follow some of the imitated motions, but fail in the majority of cases with between 9% and 32% of the episodes fully successful. The HSR is completely unable to follow these motions.

The geometric modulation approach *PR2_gm* is not able to significantly improve further upon the baseline, indicating that the velocities of the baseline that serve as input are already removing most of the difficulty from the task. Failure cases for the geometric modulation include when the starting pose lies outside the approximated reachability range as well as situations in which the EE comes close to the edges of the approximation.

In contrast, our approach to directly learn the base velocities and kinematic feasibility achieves high success rates across all robots and tasks, solving the goal reaching task with 90.0% for PR2, 71.6% for TIAGo, and 75.2% for HSR. This translates into near perfect performance on the pick&place task with 97.0% success for PR2, 91.4% for TIAGo and 90.2% for HSR. Looking at the imitation tasks, we find that these results also generalise to the unseen motions with all platforms achieving 90.6% or more of all episodes without a single kinematic failure. This indicates a number of things. First, our training task is comparably difficult, making it a good training ground to train for general tasks. This is expected as the goal distribution encompasses the full range of possible heights in which the kinematics can become quite restrictive. Focusing on a workspace restricted to common heights (*ggr restr*), performance increases for both TIAGo (+8.8%) and HSR (+4.9%) while remaining relatively unchanged for the PR2 (-1.4%). Secondly, we find no drop in performance while following the unseen imitation learning system which demonstrates the agent's ability to enable general movements of the end-effector.

We observe that the performance improves further if we consider all episodes that never deviate more than 5 cm from the desired motions (values in brackets) for both the PR2 and TIAGo. The performance drops slightly for the HSR as this is a tighter measure than the leeway that we grant its IK solutions as discussed in Sec. IV-A1. At the same time, the overwhelming majority of episodes does not use most of this leeway, with differences in the two metrics of only between 2.0% and 7.5% across all tasks. On average, the distance to the exact desired EE-motion is only 1.6 cm across all the successful steps.

Qualitatively, we find that the resulting policies take reasonable and practical paths. These include behaviours such as rotating around the closest angle, seeking out robust relative EE-poses, e.g. the PR2 moving sidewards with the right arm in front towards the goal such that the arm has the most freedoms, or if the EE pose is on the opposite side of robot body the learned strategy is to first dodge sideways to bring the end-effector in front of itself. Examples are depicted in Fig. 5 and in the accompanying video. To qualitatively examine the diversity of the learned policies, in figure 6 we plot the work space area covered by the end-effector for the PR2 agent. We find that the agent does not decay to singular behaviours, but rather uses a large area of the possible EE-poses.

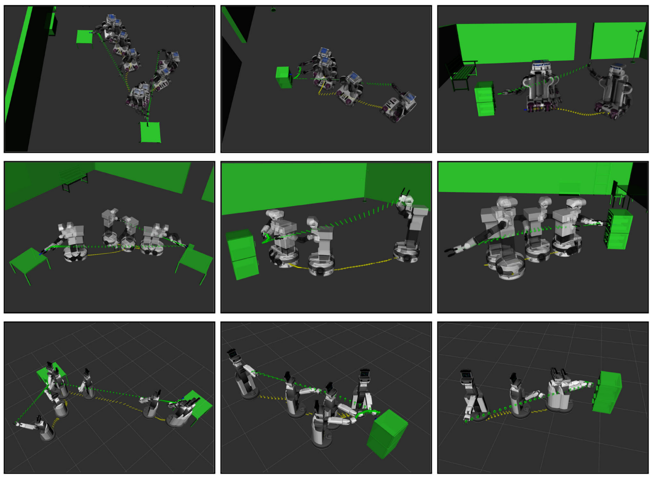


Fig. 5. Analytical evaluation on the pick&place (left), door (mid) and drawer opening (right) tasks for the PR2 (top), TIAGo (mid) and the HSR (bottom). Markers show the base (yellow) and EE-trajectory (green).

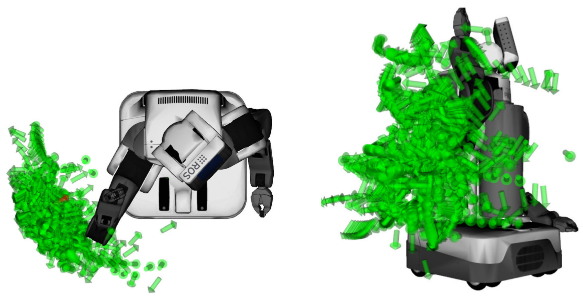


Fig. 6. Covered work space area of the learned PR2 agent: relative end-effector poses encountered over 50 episodes of the *ggr* task, evenly subsampled to 1000 poses. Red indicates kinematically infeasible poses.

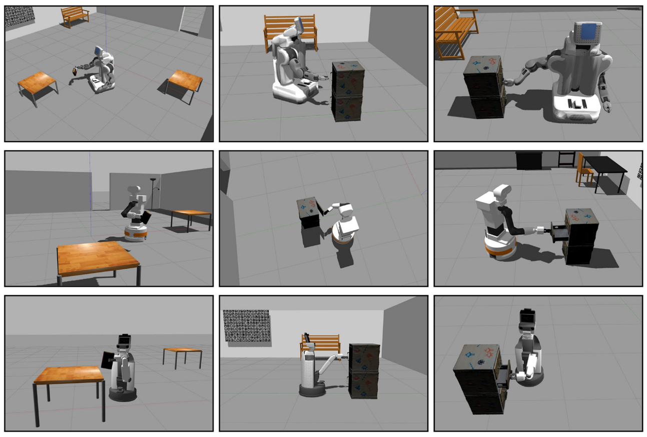


Fig. 7. Example trajectories in the Gazebo physics simulator on the pick&place (left), door (mid) and drawer opening (right) tasks for the PR2 (top), TIAGo (mid) and the HSR (bottom) robots.

C. Motion Execution in Simulation

We further analyse how well the generated motions can be executed by running the system in the Gazebo simulator [25]. This includes both a simulation of the robot controllers as well as a physical simulation of the environment. We run all the agents at a frequency of 50 Hz . By still letting the agent observe the EE-velocities $\mathbf{v}_{\mathbf{ee}}$ planned for a next timestep of 10 Hz, we can easily vary the frequency of the control loop without having to adapt the agent. The results across all approaches and tasks are summarized in Table IV and visualized in Fig. 7. Differences to the analytical environment include previously potentially unmodelled accelerations, inertias and constraints, execution time of the arm movements as well as collisions with the physical objects. We do not tune any parameters of the low-level controllers to avoid to introduce any expert knowledge. We find particularly large inertias for the simulated PR2 to cause the EE to outrun the base. To account for this, we slow down the EE-motions by a factor of two for all PR2 experiments in this section. Note that this was not needed in the real world experiments in Sec. IV-D.

TABLE IV
PERFORMANCE IN THE GAZEBO PHYSICS SIMULATOR AS SHARE OF SUCCESSFULLY EXECUTED EPISODES WITH ZERO KINEMATICALLY INFEASIBLE EE-POSES AND SHARE OF EPISODES THAT NEVER DEVIATE MORE THAN 5 CM FROM THE EE-MOTION (IN BRACKETS). THE BOLD ROWS REPRESENT OUR PROPOSED APPROACH AND TWO MODIFICATIONS THEREOF FOR IDENTIFICATION OF ERROR SOURCES

Agent	Linear Dynamic System			Imitation Learning	
	ggr	ggr restr	pick&place	door	drawer
PR2_bl	48.2 (53.0)	56.6 (62.2)	35.4 (41.2)	3.6 (4.8)	3.2 (4.2)
PR2_gm	17.0 (22.2)	19.4 (22.8)	20.8 (24.6)	21.2 (31.4)	0.1 (4.8)
PR2	**80.2 (83.6)**	**84.4 (88.8)**	**85.6 (88.8)**	**88.0 (92.8)**	**85.4 (91.6)**
PR2_hs	**87.0 (88.0)**	**89.0 (90.8)**	**92.0 (93.6)**	**94.2 (95.0)**	**90.4 (90.6)**
PR2_nObj	-	-	**87.6 (92.4)**	**85.6 (90.0)**	**85.4 (90.8)**
Tiago_bl	7.0 (7.8)	9.0 (11.0)	2.2 (2.8)	1.6 (3.0)	6.8 (7.8)
Tiago	**65.4 (70.4)**	**74.6 (81.2)**	**88.2 (91.2)**	**81.6 (86.8)**	**83.8 (88.0)**
Tiago_nObj	-	-	**87.4 (91.0)**	**93.4 (95.4)**	**92.8 (94.2)**
HSR_bl	2.6 (0.0)	2.6 (0.1)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
HSR	**64.4 (54.0)**	**70.3 (59.4)**	**85.6 (69.6)**	**87.2 (78.0)**	**90.2 (83.8)**
HSR_nObj	-	-	**92.2 (78.2)**	**87.2 (76.6)**	**87.4 (82.4)**

While all approaches experience a drop in performance compared to the analytical environment, we find this drop to be relatively small for the learned base motions, showing that they can be readily executed on all platforms. The average difference in performance across all tasks is 8.4% for PR2, 8.0% for TIAGo and 3.0% for HSR. We identify two main causes for this:

Obstacles. A main limitation of the current approach is that the base agent does not take the obstacles into account. To measure the impact of collisions with the physical objects, we execute the same motions with the objects removed from the scene, labelled *nObj* in Table IV. We find that this explains the largest part of the difference to the analytical environment for TIAGo, while the PR2 and HSR were not affected strongly by collisions. The gap to the analytical environment reduces to an average of 7.1% for PR2, 2.6% for TIAGo and 2.8% for HSR. On average, the performance with physical objects is 2.3% lower across all the tasks, further indicating that the learned base motions avoid unnecessary movements or rotations. Nonetheless, obstacle avoidance is an important piece to enable a wider variety of task setups. We aim to extend our formulation to incorporate this in future work.

Inertia. For the PR2 we find that inertias in the base movement often cause failures in the beginning of the motion when the arm starts in an already stretched out position and then wants to move further away from the base. We extend our model to give the PR2 a “head start” of three seconds to position itself before we begin the EE-motion (*PR2_hs*). To do so, we use the same trained model, i.e. the agent did not see this head start during training. This simple adaptation closes the gap to the analytical environment to 2.6%. Alternative approaches to account for large inertias would be to include acceleration limits in the linear motion system or to learn a scaling term to allow the agent to influence the EE-velocities.

The performance again further improves if we instead measure the share of episodes deviating less than 5 cm from the desired motions for both PR2 and TIAGo and the performance only slightly drops for the HSR.

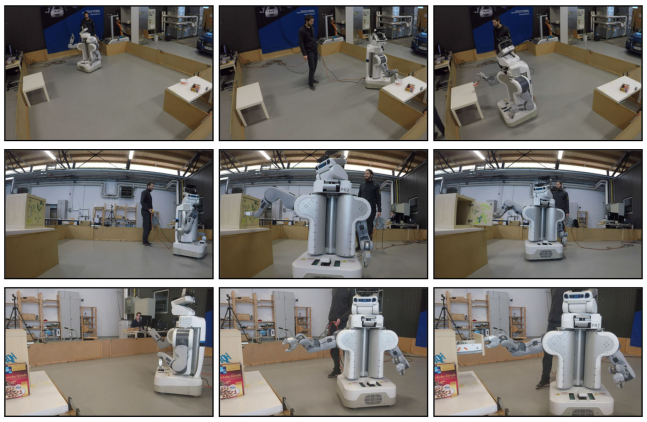


Fig. 8. Snapshots of action execution on the PR2 robot. Given the position of the manipulated objects the robot performs the *pick&place*, *grasp and open a cabinet door* and *grasp and open a drawer* tasks (top to bottom).

TABLE V
PERFORMANCE ON THE REAL WORLD PR2 ROBOT AS NUMBER OF EPISODES. KINEMATIC SUCCESS REFERS TO EPISODES WITH ZERO KINEMATIC FAILURES, TASK SUCCESS TO THE COMPLETION OF THE TASK OBJECTIVE (E.G. DOOR OPENED)

PR2	Linear Dynamic System		Imitation Learning	
	ggr	pick&place	door	drawer
Kinematic success	46	45	41	43
Task success	-	48	43	40
Total episodes	50	50	50	50

D. Motion Execution in Real World Setting

To further demonstrate the applicability of our approach in real world settings, we evaluate all the described tasks on a PR2. We construct a small environment of roughly 4.5 m $\times$6 m which is shown in Fig. 8. The PR2 uses its base scanner and a standard implementation of Adaptive Monte Carlo Localization (AMCL) to localize itself in a static map. The poses of the target objects in map frame are provided manually. To account for space constraints, we make the following adaptations: (i) we start each episode of the *general goal reaching* task from the last achieved base pose and only sample a new arm joint configuration; (ii) we reject random goals that lie outside the designated map minus a safety distance from the edges, resulting in a maximum distance of five meters between consecutive goals; (iii) in *pick&place*, we randomise the starting pose and drop off location but leave the pickup location fixed.

As in the analytical and simulated experiments, we only count executions without a single failure as kinematic success. This may include episodes where the grasp of the manipulated object fails (this does not lead to interruption). Additionally, we report episodes as task success if the manipulation is executed as desired. This may include episodes with up to 99 kinematic failures at which point we interrupt task execution. This corresponds to roughly two seconds of unsuitable configurations (50 Hz control). The results are shown in Table V and exemplary snapshots from the task execution are shown in Fig. 8. We achieve good overall results with an average of 87.5% kinematic success in the task execution. All episodes were completed without a collision or human intervention such as emergency stops. This aligns with the results from Gazebo, further indicating that the agent learns to avoid unnecessarily extensive base movements. Nonetheless we identified several sources of error for further improvement:

Controller Issues. Throughout all experiments we control the motion of robot arm and base independently. In phases of fast base rotation the arm controller sometimes lags behind in compensating the rotary motion of the base. This can lead to grasp failures or cause kinematic failures in subsequent time steps. However, this is a result of the independence of the low level base and arm controllers and not caused by our approach. A combined controller for robot arm and base could be integrated without requiring any adaptations.

Unusual Starting Configurations. Most kinematic failures occur at beginning of trajectories and are caused by difficult starting configurations and the restrictive EE-motion generator. There are very few kinematic failures during manipulation itself, i.e. when the arm is in a typical working configuration.

Configuration Jumps. Occasionally the solutions found by the IK-solver jump to very different arm joint configurations from one step to the next. These can lead to grasp failure and kinematic problems as it causes the gripper motion to deviate from the desired trajectory as the configuration shift is not instantaneous as in analytical training.

Overall the policy shows to be robust to real-world noise and inertia as well as control reaction times. Demonstrating further that the base policy learns to seek robust behaviours and positioning and successfully transfers to real-world settings in a zero-shot manner.

V. Conclusions

In this paper, we presented an approach to generate suitable robot base motions for mobile platforms given an end-effector motion generated by an arbitrary system. We formulated the problem as a reinforcement learning setting in which the robot configuration, the end-effector velocity and goal serve as observations and the robot base velocities are the corresponding actions. The environment reward is derived from the kinematic feasibility of the resulting robot base and end-effector poses. Leveraging state-of-the-art RL methods we achieve high success rates across different robot platforms for both seen and unseen end-effector motions. This demonstrates the potential of this approach to enable the application of any system that generates task specific end-effector motions across a diverse set of mobile manipulation platforms.

While we achieve very good results in terms of kinematic feasibility during trajectory generation, the approach so far does not consider collisions with the environment. In future work, we plan to incorporate obstacles and object detection into the training to enable the agent to also avoid collision while moving the base. Furthermore, we observed occasional undesired configuration jumps in the arms with high degrees of freedom which we aim to avoid by incentivising smooth joint movements through the reward function. In order to improve the performance in real world deployment, we will investigate the use of a shared controller for base and arm that will allow a faster and smoother compensation of the arm towards motion of the base. In addition to yielding a higher accuracy, this will also allow for faster execution.

References

[1] J. A. Bagnell *et al.*, "An integrated system for autonomous robotics manipulation," in *Proc. Int. Conf. Intell. Robots Syst.*, 2012, pp. 2955–2962.

[2] K. Blomqvist *et al.*, "Go fetch: Mobile manipulation in unstructured environments," 2020, *arXiv:2004.00899*.

[3] J. Stückler, M. Schwarz, and S. Behnke, "Mobile manipulation, tool use, and intuitive interaction for cognitive service robot cosero," *Front. Robot. AI*, vol. 3, no. 1, p. 58, 2016.

[4] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese, "Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation," in *Proc. Int. Conf. Robot. Automat.*, 2021.

[5] F. Paus, P. Kaiser, N. Vahrenkamp, and T. Asfour, "A combined approach for robot placement and coverage path planning for mobile manipulation," in *Proc. Int. Conf. Intell. Robots Syst.*, 2017, pp. 6285–6292.

[6] F. Burget, A. Hornung, and M. Bennewitz, "Whole-body motion planning for manipulation of articulated objects," in *Proc. Int. Conf. Robot. Automat.*, 2013, pp. 1656–1662.

[7] T. Welschehold, C. Dornhege, F. Paus, T. Asfour, and W. Burgard, "Coupling mobile base and end-effector motion in task space," in *Proc. Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.

[8] F. Burget, M. Bennewitz, and W. Burgard, "Bi2rrt*: An efficient sampling-based path planning framework for task-constrained mobile manipulation," in *Proc. Int. Conf. Intell. Robots Syst.*, 2016, pp. 3714–3721.

[9] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Robot placement based on reachability inversion," in *Proc. Int. Conf. Robot. Automat.*, 2013, pp. 1970–1975.

[10] D. Leidner, A. Dietrich, F. Schmidt, C. Borst, and A. Albu-Schäffer, "Object-centered hybrid reasoning for whole-body mobile manipulation," in *Proc. Int. Conf. Robot. Automat.*, 2014, pp. 1828–1835.

[11] J. Kindle, F. Furrer, T. Novkovic, J. J. Chung, R. Siegwart, and J. Nieto, "Whole-body control of a mobile manipulator using end-to-end reinforcement learning," 2020, *arXiv:2003.02637*.

[12] C. Wang *et al.*, "Learning mobile manipulation through deep reinforcement learning," *Sensors*, vol. 20, no. 3, p. 939, 02 2020.

[13] S. Otte, A. Zwiener, R. Hanten, and A. Zell, "Inverse recurrent models - an application scenario for many-joint robot arm control," in *Proc. Artif. Neural Netw. Mach. Learn.*, 2016, pp. 149–157.

[14] Z. Guo, J. Huang, W. Ren, and C. Wang, "A reinforcement learning approach for inverse kinematics of arm robot," in *Proc. Int. Conf. Robot. Automat.*, 2019, pp. 95–99.

[15] L. P. Kaelbling, "Learning to achieve goals," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, 1993, pp. 1094–1098.

[16] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, 2015, pp. 1312–1320.

[17] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, 1999.

[18] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 1726–1734.

[19] L. P. Kaelbling, "Hierarchical learning in stochastic domains: Preliminary results," in *Proc. 10th Int. Conf. Mach. Learn.*, vol. 951, 1993, pp. 167–173.

[20] C. Li, F. Xia, R. Martin, and S. Savarese, "Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators," in *Proc. Conf. Robot Learn.*, 2019, pp. 603–616.

[21] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 1587–1596.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 1861–1870.

[23] T. Welschehold, C. Dornhege, and W. Burgard, "Learning mobile manipulation actions from human demonstrations," in *Proc. Int. Conf. Intell. Robots Syst.*, 2017, pp. 3196–3201.

[24] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," 2019, Accessed: 8.7.2021. [Online]. Available: https://github.com/DLR-RM/stable-baselines3.

[25] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2004, pp. 2149–2154.

D

AgroRL: End-to-end Path Planning of Air-Ground Multi-Robot Team for Green Digital Farming

Halil Ibrahim Ugurlu, Deniz Bardakci, Huy Xuan Pham and Erdal Kayacan

***Abstract*— Increasing the operational efficiency of agricultural machines is essential by the use of artificial intelligence (AI)-based navigation, planning, and control algorithms to handle the increasing demand for food production without compromising sustainability. In this study, a novel end-to-end path planning algorithm (AgroRL) is proposed for multiple aerial-ground robots team for green transition in agriculture. In the proposed solution, while main operations in the field are handled by the ground vehicle, the aerial robot is responsible for re-planning a collision-free trajectory for the ground robot when the robot faces an obstacle. Deep reinforcement learning is used for training the end-to-end policy for local re-planning of the aerial robot. The agent, informed by the global trajectory, generates local plans based on depth images. Variational autoencoders are also investigated for dimension reduction of the depth images in obstacle avoidance context to speed up deep reinforcement learning and alleviate the computational complexity of the policy network. The agriculture environment is developed in the Webots open-source robot simulator. Finally, the efficiency and efficacy of the ground-aerial robot team are evaluated over a number of cluttered field scenarios. The extensive simulation and real-world experiments demonstrate that the use of an aerial robot enhances the ground robot's capabilities significantly compared to having further sensors on the ground robot.**[1]

I. Introduction

Robot teams are commonly used for their ability to work on a single task collaboratively. They are especially advantageous when the robots have different capabilities and/or operate in different domains. For instance, a collaboration between an aerial and a ground robot might be helpful: long-term and close contact tasks are assigned to the ground robot due to its relatively higher power source, and observation tasks are assigned to the aerial vehicle due to its better field of view.

In this study, a collaborative solution is proposed in an agricultural field to leverage the efficiency of the agricultural operation with artificial intelligence (AI)-based navigation methods. Particularly, unmanned ground vehicles (UGVs) are responsible for field operations, *e.g.*, seeding, and an unmanned aerial vehicle (UAV) deals with the navigation performance enhancement of the team. Since machinery and equipment are the major costs in agricultural operations and ground vehicles (*e.g.*, a tractor costs around 200K USD) are significantly more expensive than the aerial robots (*e.g.*, a typical mid-size UAV costs 10K USD), the motivation behind this study is to reduce the overall cost of the operation by also increasing the accuracy and productivity. In order to minimize the cost of the ground vehicle, the UGV is considered blind in its contribution to the planning task and accomplishes its field tasks with the given motion commands. This work focuses on the motion planning of the robot team using the information of a global path to follow and the depth camera on the UAV. In case of an obstacle occurrence on the global path, the UAV follows a collision-free path and informs the UGV about the required maneuver. In this way, the required sensor costs on the UGV is minimized.

State estimation, perception, planning, and control are conventionally considered as separate problems to be solved in autonomous robot navigation. On the other hand, recent developments in machine learning, particularly in deep reinforcement learning (DRL), enable an agent to learn various navigation tasks with only a single neural network policy. These methods are promising to solve navigation problems faster since they do not deal with the unnecessary optimization of particular problems; however, they are also hard to debug, making them hard to apply in real scenarios. In this work, as an end-to-end planning method, a policy network for collaborative agricultural planning is trained using DRL, which provides position steps using a multi-modal input, a depth camera, and global trajectory information.

While DRL methods allow solving sequential decision-making tasks by only defining a reward function, they are considered sample inefficient. They need enormous amount of training samples in order to reach a satisfactory performance. The sample inefficiency rises with the dimension of state or action spaces in the problem definition. An option to reduce the dimensionality is to encode -i.e., using a variational autoencoder (VAE)- high-dimensional data, such as the depth image. Therefore, the effect of encoding the depth image is investigated to reduce the load on the DRL algorithm and to obtain a faster policy in this study.

A. Contributions

The contributions of this paper are fourfold:

- We propose an open-source RL framework (AgroRL) for training an end-to-end planner for a quadrotor UAV.
- We investigate VAE-based state representation for end-to-end reactive planning of UAVs.
- We integrate multiple UGVs working in an agricultural field with the UAV agent to generate collision-free local motion plans.

H.I. Ugurlu, D. Bardakci, H.X. Pham, E. Kayacan are with Artificial Intelligence in Robotics Laboratory (Air Lab), the Department of Electrical and Computer Engineering, Aarhus University, 8000 Aarhus C, Denmark (email: {halil, huy.pham, erdal} at ece.au.dk; deniz-bardakci at outlook.com)

[1]The code, trained models and simulation environment will be available at https://github.com/open-airlab/agro_rl

- The method is evaluated with extensive experiments in a Webots-based simulation environment and demonstrated in a real-world indoor scenario.

B. Related work

As a machine learning paradigm, reinforcement learning (RL) aims to solve sequential decision-making problems through the interaction of a learning agent with its environment [1]. With the success of the deep learning models in machine learning, it is also applied with RL, which brings DRL field with success in several benchmark problems such as video games [2] or continuous control tasks [3]. Several methods are proposed to optimize deep neural networks to learn the value function [2], policy function [4], or both [3], [5] in the RL domain, such as the proximal policy optimization (PPO) [6] algorithm, a state-of-the-art method utilized in this work. Nevertheless, model-free DRL methods are sample inefficient, especially when applied to high dimensional observations such as pixels [7]. One approach in the literature to cure this issue is to apply representation learning to provide a better representation of the observation to the policy network using auto-encoders [8], data augmentation [9], or contrastive learning [10]. This work has investigated the effects of a VAE-based [11] latent representation of the images on the learning performance.

These developments in DRL are applied to robotics applications, including real-world demonstrations. Specifically, neural network policies are trained to control quadrotor UAVs in attitude [12] or position [13], [14] level using full state information. They are also applied to navigate a UAV with visual information such as corridor following [15], drone racing [16], or obstacle avoidance [17].

With the increasing demand to the using agricultural resources efficiently and sustainably, the need for involvement of the new technologies has become an inevitable fact. As a result, aerial robots have been adopted for various tasks, such as spraying [18], environment mapping [19], monitoring [20], [21], remote sensing [22], and many more, to achieve this ambitious goal. In addition to UAVs, ground vehicles are also getting more autonomy in the production phases starting from soil preparation to crop harvesting with the implementation of both learning-based [23] and model-based [24]–[27] advanced methods. After all, multi-robot UAV-UGV cooperative systems are widely studied in agricultural robotics for sensor planning [28], collaborative mapping [29], monitoring [30], and handling sensory faults [31]. In this work, a depth camera-equipped UAV is considered to help a team of UGVs for obstacle-free path planning.

The remainder of this paper is organized as follows. Section II explains the end-to-end planning methodology for a quadrotor UAV with the formalization of the RL problem. The section also delivers the details of the local replanning and control of UGVs in the agricultural scenario. Section III provides the experimental setup and the comprehensive tests of the proposed method in the simulation environment. The section also provides the results of the real-time tests. Finally, Section IV concludes this work with future research directions.

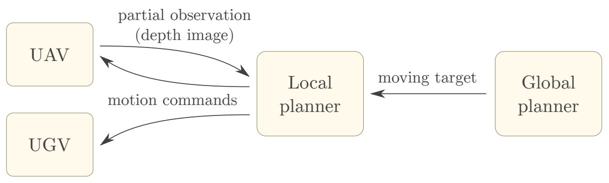


Fig. 1: The aerial-ground robot team is commanded by handling the partial observation acquired with the depth camera on the UAV. The team follows the global trajectory while avoiding obstacles.

II. End-to-end motion planning of UAV and UGVs

A. Problem definition and assumptions

The general block diagram of the methodology is given in Fig. 1. The local planner is informed about the global plan as a moving target. The local planner incorporates the moving target together with the current depth image and generates local plans for UAV and UGV. Therefore, it helps the UAV-UGV team to track the globally defined trajectory while avoiding obstacles. The local planner is implemented as a learning-based end-to-end motion planner.

B. Reinforcement learning formalization of the environment

An RL problem is generally formalized as a Markov decision process (MDP) with state, action, and reward components with discrete time steps, t. For the problem of end-to-end agricultural local planning, the state is multi-modal data containing the depth image and the vector representing the moving target point, defined as,

$$s_t = (I_{depth,t}, \mathbf{p}_t), \tag{1}$$

where $I_{depth,t}$ is 64×64 matrix representing depth image and $\mathbf{p}_t = [x_t, y_t]^T$ is 2×1 vector representing the position of the target point with respect to the body frame.

There are seven actions defined as a combination of $1\mathrm{m}$ position step in three possible directions and a turn in yaw angle with respect to the drone's reference frame, as shown in Table I. Possible actions are also illustrated in Fig. 3a. The position step direction is kept small in order to fit with the field-of-view (FOV) of the depth camera so that the UAV does not hit an unseen object. On the other hand, yaw angle change enables a sharper turn around an obstacle as well as a change of point of view if required. An episode is started when the UAV is at the beginning of a route. At each timestep, an action is applied to the UAV, then the depth image and next target point are obtained as the new state. Fig. 2 illustrates the selection of the target point projected on the global path and $5\mathrm{m}$ ahead of the drone for consecutive time steps. The episode is finalized under three conditions: crashing an obstacle, deviating from the global trajectory, and finalizing the route.

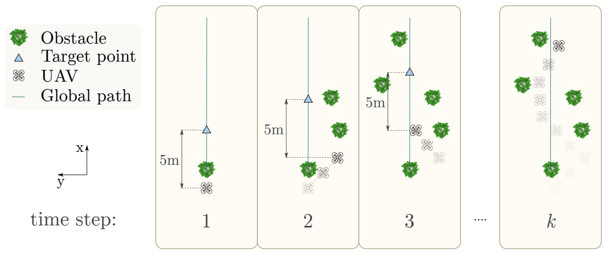


Fig. 2: Moving target generation for the end-to-end agent. The target point is located 5m in advance of the current location of the UAV projected onto the global path. The trees (obstacles), generated target points, UAV, and the global path are represented as shown in the legend. The UAV takes a position step at each time step and is informed with the vector showing the generated target point. The overall trajectory is demonstrated at the k'th time step.

The reward signal is based on the UAV's relative motion at every time step if the episode is not terminated. For termination of an episode, the collision and excessive deviation are punished with constant values. On the other hand, finishing a route normally is rewarded. The reward signal is defined as,

$$r_t = \begin{cases} 2\Delta x - dy + 0.3d\theta, & \text{for non-terminal steps} \\ R_{dp}, & \text{for } dy > 5\text{m}, \\ R_{cp}, & \text{for collision}, \\ R_{fr}, & \text{for finishing normally} \end{cases} \quad (2)$$

where Δx, dy and $d\theta$ are the distance traveled forward, the distance to the global trajectory and yaw angle difference from forward looking, $R_{dp} = -10$, $R_{cp} = -50$ and $R_{fr} = 20$ are punishment for excessive deviation, punishment for collision, and reward for finishing an episode without any crash. This reward logic enables the agent to avoid obstacles while quickly navigating to goal.

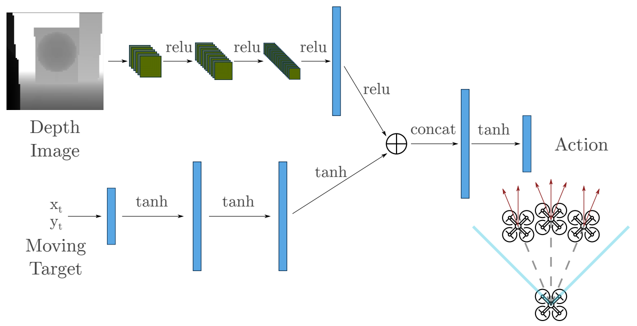
C. Policy network architecture

The policy network ($\pi(s_t)$) structure for this task is given in Fig. 3a. The neural network feeds the depth image to convolutional layers and the vector of moving target position to fully connected layers. The two networks are concatenated to fully connected layers, and the action is generated. The algorithm trains this policy network with given settings using a DRL method, PPO [6].

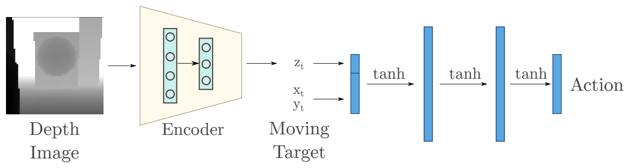
Additionally, we have trained a VAE to have a latent representation of the depth image. As shown in Fig. 3b, this representation is appended with the moving target and fed to a simpler policy network. Such a framework makes the overall network more lightweight than the vanilla setting. We have collected around 10^5 depth images with obstacles around the environment to train the VAE.

TABLE I: Each action is applied as a position step and a turn in heading angle with respect to the drone's reference frame.

Choice	Position step (x, y)	Heading angle
Action 1	$(\cos(22.5°)\text{m}, \sin(22.5°)\text{m})$	$22.5°$
Action 2	$(\cos(22.5°)\text{m}, \sin(22.5°)\text{m})$	$0°$
Action 3	$(1\text{m}, 0\text{m})$	$22.5°$
Action 4	$(1\text{m}, 0\text{m})$	$0°$
Action 5	$(1\text{m}, 0\text{m})$	$-22.5°$
Action 6	$(\cos(22.5°)\text{m}, -\sin(22.5°)\text{m})$	$0°$
Action 7	$(\cos(22.5°)\text{m}, -\sin(22.5°)\text{m})$	$-22.5°$



(a) Policy network structure of the end-to-end local planner. The actions of the end-to-end planning agent are illustrated at the output of the network, where the FOV, position steps, and heading angle are presented as blue lines, dashed lines, and red arrows, respectively.



(b) Policy network structure of end-to-end local planner with encoded depth image. The latent representation of the depth image is concatenated with the moving target. The neural network with two hidden layers is trained with DRL.

Fig. 3: Policy network structures

D. Local re-planning for UGV

The path generated by the UAV is employed for the path planning of the UGV as presented in Algorithm 1. The algorithm uses the waypoints of the UAV in a list, $path_{uav}$, and the y-axis value that the global trajectory lies on, y_{global}. The waypoints are considered on the global trajectory within a certain threshold, ± 0.4. When the deviation from the global path exceeds the threshold, the waypoints are shifted to the right or left accordingly to provide a safe target point for the UGV as the body of the UGV is bigger than the UAV. The target point, (p_x, p_y), is passed to the controller of the UGV. If the distance to the target point decreased under a threshold waypoint counter, p_{count}, is increased by two. Therefore two consecutive waypoints provided to the UGV are around 2m apart due to the size of the UGV deployed.

The controller of the UGV is presented in Algorithm 2, which commands the steering angle, θ_c, and the cruising speed, v_c, to the tractor UGV. Proportional controllers with saturation are implemented to reduce both yaw angle error, e_θ, and the distance error, e_d, of the UGV. The steering angle command is passed through a discrete low-pass filter to prevent the tractor from falling due to a sharp turn. The settling time of the filter is adjusted around 1s with the frequency of the controller 60Hz.

III. Experiments and Results

This section explains the simulation setup to train and test the end-to-end planning agent, the conducted collaborative work experiments of the UAV and the UGV in a cluttered agriculture field, and real-time test results in the laboratory environment (Artificial Intelligence in Robotics Laboratory at Aarhus University, Denmark).

A. Simulation setup

A Webots-based simulation environment has been developed to train and test the proposed algorithms (See Fig. 4). Webots [32] is an open-source three-dimensional mobile robot simulator, which allows different programming interfaces, such as C/C++, python, or ROS, several kinds of robots in various application fields. The 2021a release of Webots has been utilized to develop the cluttered agriculture environment and deploy the UAV and UGVs with the required sensory equipment. A third-party software package, ArduPilot, is selected to implement a quadrotor UAV in Webots to benefit from its MAVLink extendable communication featured stable and reliable UAV with its Webots SITL extension. The UAV robot is then equipped with a depth camera to provide the required information to carry end-to-end planning operations. The environment needed to be reset every time collision occurs in training which is a benefit we can have in simulation throughout the trial and error process.

The simulation environment is wrapped as an OpenAI gym environment [33] to allow the required communication between the DRL algorithm and the environment. The robot operating system (ROS) [34] handles this communication between the gym wrapper and the simulation. Specifically, the MAVROS package is used to acquire the global positioning system (GPS) location of the quadrotor UAV and send position commands. The remaining information, such as depth image, collision, UGV position, and UGV steering commands, is communicated directly by individual Webots ROS topics. The gym environment interfaces the simulation environment as an MDP for the DRL algorithm, as explained in Section II-B.

Algorithm 1 UGV planner

```
Require: $path_{uav}[\,], y_{global}$
    $p_{count} \leftarrow 0$
    while $p_{count} < path\_size$ do
        $p_x \leftarrow path_{uav}[p_{count}].x$        ▷ Get the next waypoint.
        $p_y \leftarrow path_{uav}[p_{count}].y$
        if $p_y - y_{global} < -0.4$ then
            $p_y \leftarrow p_y - 0.5$        ▷ Extend point to the right.
        else if $p_y - y_{global} > 0.4$ then
            $p_y \leftarrow p_y + 0.5$        ▷ Extend point to the left.
        else
            $p_y \leftarrow y_{global}$        ▷ Keep point on the global path.
        end if
        $e_d \leftarrow$ UGV_CONTROLLER($p_x, p_y$)
        if $e_d < 0.2$ then
            $p_{count} \leftarrow p_{count} + 2$
        end if
    end while
```

Algorithm 2 UGV controller

```
function UGV_CONTROLLER($p_x, p_y$)
    $e_\theta, e_d \leftarrow$ CALCULATE_ERRORS($p_x, p_y$)
    $\theta'_c \leftarrow \min(\max(-K_\theta e_\theta, -0.65), 0.65)$
    $\theta_c \leftarrow 0.05(\theta'_c - \theta_c) + \theta_c$
    $v_c \leftarrow \min(K_d e_d, 0.4)$
    COMMAND_TRACTOR($v_c, \theta_c$)
    return $e_d$
end function
```

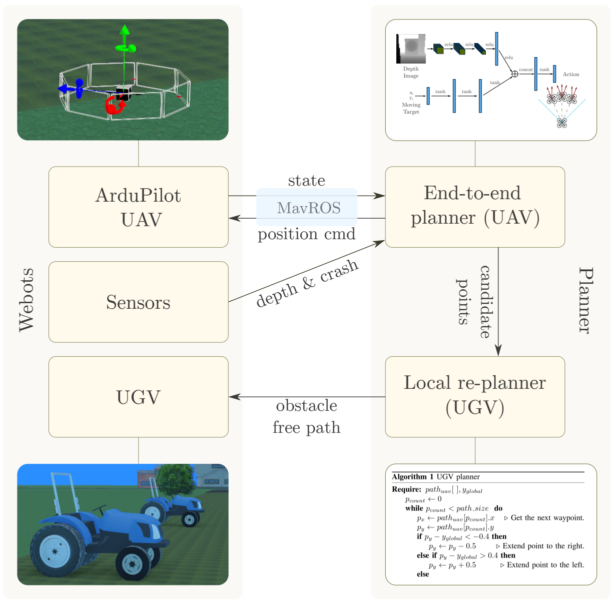


Fig. 4: Overall block diagram of the collaborative operations. A single UAV is generating obstacle-free way points by deploying the end-to-end planner network to guide multiple UGVs in the agricultural field.

B. Training in simulation

The end-to-end planner is trained in Webots with seven routes to present a variety of data for the deep network, as given in Fig. 5a. The agent is subject to different obstacle shapes and different kinds of obstacle densities starting from no obstacle. That enables the agent to generalize the experience during RL training. Each episode begins on a randomly selected route and terminates either at the end of the route or in a collision.

TABLE II: Completion time (in seconds), travel distance (in meters), and success rate of methods over 20 runs at 10 test routes. The completion time is calculated over fully completed tracks and given as mean and standard deviation. The travel distance is given as mean and standard deviation.

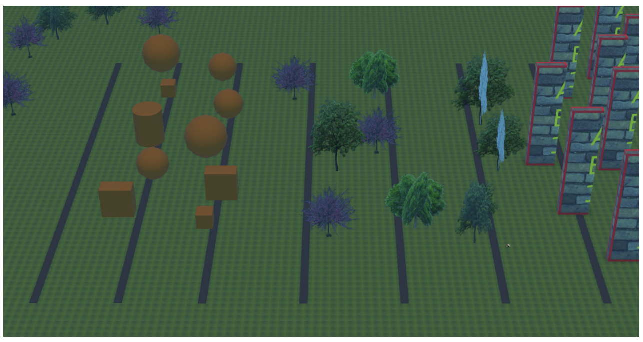
Environment:		Route 1	Route 2	Route 3	Route 4	Route 5	Route 6	Route 7	Route 8	Route 9	Route 10	Overall
PF	**time**	136.5 ± 2.3	132.0 ± 2.5	134.3 ± 1.1	130.5 ± 1.1	132.8 ± 1.9	130.6 ± 0.0	134.2 ± 1.6	135.2 ± 1.1	135.5 ± 0.5	134.4 ± 0.0	133.8 ± 2.7
	distance	30.0 ± 0.0	24.14 ± 8.0	25.81 ± 4.2	27.4 ± 3.2	24.1 ± 6.7	26.75 ± 0.9	26.52 ± 7.2	10.56 ± 9.7	18.06 ± 8.5	14.47 ± 10.4	22.8 ± 9.1
	success rate	100%	60%	50%	60%	50%	5%	80%	20%	10%	5%	44%
AgroRL	**time**	131.8 ± 2.0	129.8 ± 1.4	128.7 ± 1.6	126.5 ± 1.0	130.4 ± 1.3	127.5 ± 1.1	130.1 ± 1.2	130.2 ± 1.8	132.2 ± 1.5	128.6 ± 1.7	129.4 ± 2.2
	distance	27.59 ± 7.2	27.71 ± 5.1	30.0 ± 0.0	30.0 ± 0.0	17.61 ± 9.0	24.48 ± 6.3	29.27 ± 3.2	30.0 ± 0.0	21.13 ± 7.7	27.64 ± 7.1	$\mathbf{26.5 \pm 6.9}$
	success rate	90%	80%	100%	100%	30%	45%	95%	100%	30%	90%	**76%**
AgroRL - $R_{cp} = -20$	**time**	133.7 ± 1.5	133.4 ± 1.7	130.7 ± 1.3	132.7 ± 3.8	137.3 ± 3.4	131.6 ± 1.7	132.3 ± 1.7	132.0 ± 2.2	130.7 ± 0.8	131.2 ± 1.4	132.5 ± 2.7
	distance	30.0 ± 0.0	17.05 ± 9.1	29.52 ± 2.1	30.0 ± 0.0	21.42 ± 10.1	23.46 ± 9.2	30.0 ± 0.0	29.49 ± 1.6	15.7 ± 9.6	27.58 ± 6.9	25.4 ± 8.4
	success rate	100%	30%	95%	100%	50%	60%	100%	90%	25%	85%	73.5%
AgroRL - $n_{steps} = 128$	**time**	135.0 ± 1.6	132.0 ± 1.3	131.5 ± 2.6	130.3 ± 1.3	131.5 ± 3.1	131.1 ± 1.1	131.3 ± 1.4	132.9 ± 1.7	132.1 ± 1.3	131.1 ± 0.7	131.9 ± 2.2
	distance	27.76 ± 6.7	17.85 ± 9.9	25.66 ± 4.4	30.0 ± 0.0	23.63 ± 10.4	30.0 ± 0.0	30.0 ± 0.0	30.0 ± 0.0	21.67 ± 3.9	19.28 ± 11.8	25.6 ± 7.9
	success rate	90%	40%	50%	100%	70%	100%	100%	100%	15%	55%	72.0%
AgroRL - $R_{fr} = 0$	**time**	132.4 ± 1.6	129.6 ± 1.5	130.8 ± 1.1	129.1 ± 1.6	127.4 ± 0.0	130.7 ± 0.7	131.4 ± 1.0	130.2 ± 1.4	-	129.8 ± 1.7	130.4 ± 1.8
	distance	27.86 ± 6.4	30.0 ± 0.0	23.77 ± 7.5	28.85 ± 5.0	14.51 ± 4.7	20.85 ± 3.9	29.12 ± 2.6	25.91 ± 6.0	17.69 ± 3.8	27.09 ± 6.9	24.6 ± 7.2
	success rate	90%	100%	55%	95%	5%	15%	90%	65%	0%	70%	58.5%
AgroRL - $l_{step} = 0.5$	**time**	190.2 ± 1.5	-	189.4 ± 1.5	187.7 ± 1.8	-	191.2 ± 0.0	190.4 ± 1.7	187.5 ± 1.2	188.5 ± 1.2	186.7 ± 1.9	188.7 ± 2.1
	distance	30.0 ± 0.0	9.65 ± 0.4	24.18 ± 4.8	23.94 ± 8.3	7.41 ± 3.7	19.83 ± 2.3	30.0 ± 0.0	29.23 ± 3.4	23.51 ± 6.4	26.86 ± 7.7	22.5 ± 9.0
	success rate	100%	0%	40%	65%	0%	5%	100%	95%	40%	85%	53.0%
AgroRL_VAE	**time**	131.9 ± 2.9	128.8 ± 1.6	143.6 ± 1.4	128.2 ± 0.6	128.3 ± 0.7	129.5 ± 1.8	-	-	128.2 ± 0.0	128.1 ± 0.9	129.8 ± 3.8
	distance	30.0 ± 0.0	25.87 ± 8.3	16.63 ± 7.9	29.68 ± 1.4	12.14 ± 10.6	14.45 ± 12.7	5.5 ± 0.2	10.47 ± 1.5	26.94 ± 5.3	7.63 ± 5.4	17.9 ± 11.2
	success rate	100%	80%	15%	95%	25%	40%	0%	0%	75%	5%	43.5%

The policy network is trained with PPO algorithm implementation in stable-baselines [35] with python 3. A custom policy is implemented for the neural network shown in Fig. 3a to receive both image and vector input. The algorithm is trained through 40000 time-steps, and the best network is stored during training based on the reward performance in the recent 20 episodes.

C. UAV tests in simulation

A different set of 10 routes has been designed to test and compare the proposed method's efficiency, as shown in Fig. 5b. The obstacles from the training phase are mixed in paths with differing sizes and densities to create the challenge to compare methods. Also, each route starts with a random offset of $\pm 0.5m$ in vertical for evaluation purposes. Each method is evaluated 20 times in each lane, and success rate, completion time, and traveled distance are listed statistically in Table II. The proposed method is reported as "AgroRL" with five versions to investigate the effect of a single parameter change on the resulting planner. The four modified options are listed with the labels "$R_{cp} = -20$", "$n_{steps} = 128$", "$R_{fr} = 0$" and "$l_{step} = 0.5$" where R_{cp} and R_{fr} are parameters in Eqn. 2, n_{steps} is the number of time steps for PPO policy update, and l_{step} is the position step length of an action. The proposed method using VAE-based representation of the depth image is reported with the label "VAE".

As a baseline method, an artificial potential field-based planner is implemented and reported as "PF" in Table II. In the implementation, each pixel in the middle row of the depth image creates a repulsive force, and the moving target creates an attractive force. So, the algorithm uses the same observation with the end-to-end planner. The action is also selected from the same action set according to the direction of the common artificial force in the reference frame of the UAV. The angle space, $[180°, -180°]$, is divided at the degrees $\{30°, 20°, 5°, -5°, -20°, -30°\}$, and an action from 1 to 7 is selected for each slot, respectively. These parameters are manually tuned on the training routes and then tested to compare with the proposed method.


(a) Seven training routes. From left to right: one no-obstacle, two with geometric-shaped hanging obstacles, three paths with tree models, one with blocks. Black lines represent the projection of a 30m global trajectory on the ground.

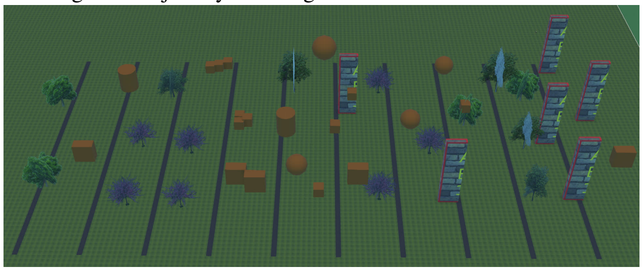
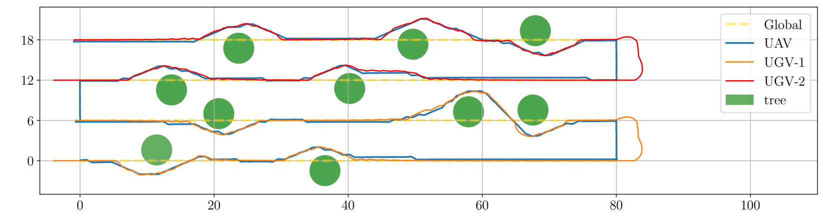
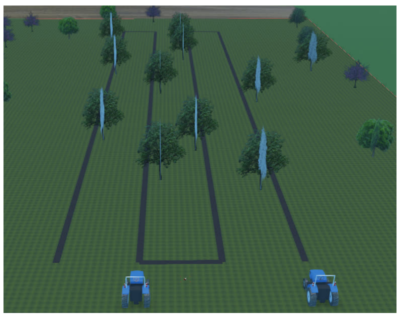

(b) Ten testing routes. The obstacles are mixed to create various levels of challenges in paths. Black lines represent the projection of a 30m global trajectory on the ground.

Fig. 5: Training and testing routes.

According to the test results, the finishing reward and the step length have a higher effect on performance when compared to the other two options. The finishing reward shows the importance of a higher level reward in the RL setting. In this case, while every step is rewarded in lower-level if the policy sticks with the global trajectory, the finishing reward allows the algorithm to evaluate the policy's performance over the whole episode, which is critical for having a safer flight. As a result, the policy will be more robust in dense obstacle environments with the finishing reward, e.g., the "AgroRL - $R_{fr} = 0$" method has a higher success rate in



(a) Global trajectory and trajectories of UAV and UGVs are plotted. The trees are represented equivalent with their radius in simulation environment.



(b) The simulated agricultural field.

Fig. 6: Multiple UGV guidance is demonstrated in the agricultural simulation track with one UAV and two UGVs/tractors. The global trajectory covers the field with forward and backward 80m length paths. While each UGV acts on a single forward and backward path, the UAV follows the whole route and generates obstacle-free plans for each UGV.

sparse routes, such as routes 1 and 2, when compared to ”AgroRL”. We have also tested a lower position step length which yields a higher resolution in the planning configuration space. However, this case completes the episodes slower and achieves a worse performance in the environment. We deduce that 1m step length is sufficient for the obstacle sizes in agricultural applications considering tree size in the field.

In the experiments with VAE, the model cannot achieve comparable performance with the vanilla case. One possible explanation is the generalization capacity of the encoder model. A single hidden layer network is implemented for the encoder and the decoder to obtain a lower computational complexity than AgroRL, which can be run 15Hz in an Nvidia Jetson TX2 board. The second explanation for the performance of AgroRL_VAE is the representation learning methodology. The encoder is trained with previously collected data from the simulation environment, and then the DRL algorithm runs on top of it. However, there are methods in the literature combining DRL with the representation learning in the same training and feeding the representation learning block with the reward information from the environment, which can result in a more accurate representation of the data.

D. UAV and multiple UGV scenario in simulation

The method is also tested in the simulated agricultural field with a UAV guiding multiple UGV tractors. An 80m long agricultural field with a global trajectory is designed going forward and backward for covering the area, as seen in Fig. 6b. Each tractor follows one forward and one backward path; however, this can be extended to a higher number of tractors. The UAV follows the entire trajectory while providing an obstacle-free path to the relative UGV. While UGVs are moving slower due to their field operations, the UAV can continue generating a new path for the next UGV. We have plotted the GPS locations of the UAV and UGVs in the given scenario in Fig. 6a, where the UAV successfully generates a path following the global trajectory while diverging from a safe distance to the trees.

E. Real time experiments

The real-world experiments demonstrate that the proposed UAV planner trained solely with simulation can directly work in a real environment. A quadrotor is deployed carrying an Intel Realsense depth camera D435i running at 30Hz, as shown in Fig. 7. The drone is controlled by a Pixhawk autopilot [36]. The overall framework runs entirely on-board on an NVIDIA Jetson TX2 computer, except that the robot’s localization is provided by a motion capture system. The

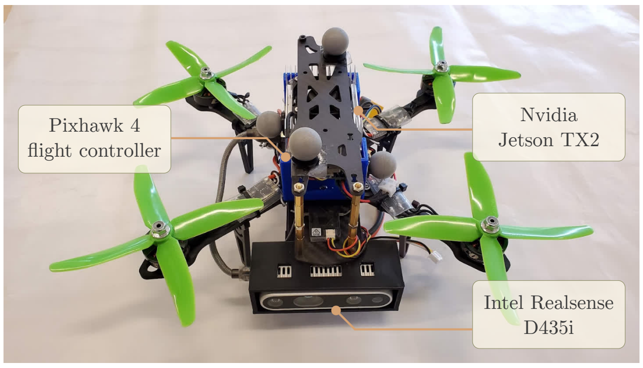
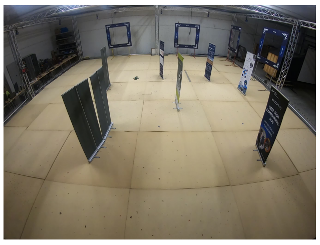
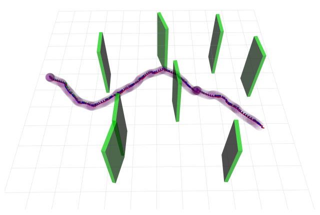


Fig. 7: Custom drone used in real-time experiments.



(a) The experimental setting with randomly placed obstacles.



(b) A resulting trajectory of the drone running AgroRL.

Fig. 8: The end-to-end planner is deployed in real-time experiments with 10m route and cluttered obstacles in the motion capture laboratory.

actions of the policy network are converted into desired poses at $15\mathrm{Hz}$ in the drone frame, and a geometric controller [37] is used to track the poses accurately.

Since the real depth images are noisy compared to simulated data, we enhance raw input images by using a fast depth dilation algorithm [38] and then resize the cropped top part of the image to 64×64 to feed the policy network. We found that processed depth images help to bridge the gap between simulation and the real world. Fig. 8 shows the trajectory[2] of our drone navigating at $2\mathrm{m/s}$ safely through a $10\mathrm{m}$ route with multiple obstacles densely placed along the global path. Similar to the simulations, the drone successfully navigates through obstacles, with the narrowest passage being approximately three times the drone size. The movements of the drone in the experiments are very jerky due to poor low-level controller tuning. This issue can be improved by using minimum-jerk trajectory generation [39] and better controller tuning and will be addressed in our future work. However, the network still performs robustly under aggressive movement noises, and the task is completed satisfactorily.

IV. Conclusion and Future Work

In this work, an end-to-end planner is trained with DRL for local replanning in agricultural use-case. An agricultural simulation environment has been developed in Webots. The end-to-end planning algorithm is trained and tested in comprehensive simulations. The guidance of multiple UGVs is also demonstrated with a single UAV deployed with the end-to-end planner. The method is also deployed in real-world indoor environment successfully.

The end-to-end planner outperforms a baseline implementation based on the artificial potential field method, which has a lower success rate, especially in cluttered obstacle settings. This shows that AgroRL has learned to make better long-term decisions. The importance of a high-level reward in DRL training is also verified by providing a reward for successfully finishing an episode to the agent where the agent shows an 18% higher success rate. One downside of the method is that it is not sufficient to deploy continuously on an onboard computer, such as NVIDIA Jetson TX2. So, the method is implemented discretely where a new position reference is provided once in a while. To decrease computational complexity, a VAE-based representation of the depth image is utilized in training. However, the performance cannot match the proposed method.

One possible future research direction is to implement a better representation learning methodology instead of decoupling representation learning and DRL. When the algorithm can run continuously in real-time, there is a possibility to provide lower-level control commands to the UAV, which yields a faster flight. Another future direction is to integrate multiple robots (UAV and UGVs) in the same learning problem. Since the end-to-end planner is separated from the UGV planner, only common obstacles can be considered in the problem. However, one advantage of having a UAV is its better FOV. Combining multi-robot planning in the same problem will provide the ability to avoid obstacles in the ground without bothering the UAV.

[2]The video of real-time experiment can be found: `https://youtu.be/dWoW464F82s`

Acknowledgment

This work is supported by the European Union's Horizon 2020 Research and Innovation Program (OpenDR) under Grant 871449. This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[7] Y. Yu, "Towards sample efficient reinforcement learning." in *IJCAI*, 2018, pp. 5739–5743.

[8] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," *arXiv preprint arXiv:1910.01741*, 2019.

[9] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *arXiv preprint arXiv:2004.14990*, 2020.

[10] A. Srinivas, M. Laskin, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," *arXiv preprint arXiv:2004.04136*, 2020.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[12] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.

[13] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 59–66.

[14] H. I. Ugurlu, S. Kalkan, and A. Saranli, "Reinforcement learning versus conventional control for controlling a planar bi-rotor platform with tail appendage," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 4, pp. 1–17, 2021.

[15] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 6008–6014.

[16] R. Bonatti, R. Madaan, V. Vineet, S. Scherer, and A. Kapoor, "Learning visuomotor policies for aerial navigation using cross-modal representations," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 1637–1644.

[17] E. Camci, D. Campolo, and E. Kayacan, "Deep reinforcement learning for motion planning of quadrotors using raw depth images," *learning (RL)*, vol. 10, p. 12, 2020.

[18] B. S. Faiçal, H. Freitas, P. H. Gomes, L. Y. Mano, G. Pessin, A. C. de Carvalho, B. Krishnamachari, and J. Ueyama, "An adaptive approach for uav-based pesticide spraying in dynamic environments," *Computers and Electronics in Agriculture*, vol. 138, pp. 210–223, 2017.

[19] W. Dong, P. Roy, and V. Isler, "Semantic mapping for orchard environments by merging two-sides reconstructions of tree rows," *Journal of Field Robotics*, vol. 37, no. 1, pp. 97–121, 2020.

[20] S. W. Chen, S. S. Shivakumar, S. Dcunha, J. Das, E. Okon, C. Qu, C. J. Taylor, and V. Kumar, "Counting apples and oranges with deep learning: A data-driven approach," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 781–788, 2017.

[21] J. Das, G. Cross, C. Qu, A. Makineni, P. Tokekar, Y. Mulgaonkar, and V. Kumar, "Devices, systems, and methods for automated monitoring enabling precision agriculture," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2015, pp. 462–469.

[22] W. H. Maes and K. Steppe, "Perspectives for remote sensing with unmanned aerial vehicles in precision agriculture," *Trends in plant science*, vol. 24, no. 2, pp. 152–164, 2019.

[23] E. Kayacan, E. Kayacan, H. Ramon, and W. Saeys, "Learning in centralized nonlinear model predictive control: Application to an autonomous tractor-trailer system," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 197–205, 2015.

[24] ——, "Distributed nonlinear model predictive control of an autonomous tractor–trailer system," *Mechatronics*, vol. 24, no. 8, pp. 926–933, 2014.

[25] T. Kraus, H. J. Ferreau, E. Kayacan, H. Ramon, J. De Baerdemaeker, M. Diehl, and W. Saeys, "Moving horizon estimation and nonlinear model predictive control for autonomous agricultural vehicles," *Computers and electronics in agriculture*, vol. 98, pp. 25–33, 2013.

[26] E. Kayacan, E. Kayacan, H. Ramon, and W. Saeys, "Robust tube-based decentralized nonlinear model predictive control of an autonomous tractor-trailer system," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 1, pp. 447–456, 2015.

[27] E. Kayacan, H. Ramon, and W. Saeys, "Robust trajectory tracking error model-based predictive control for unmanned ground vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 2, pp. 806–814, 2016.

[28] P. Tokekar, J. V. Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic uav and ugv system for precision agriculture," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1498–1511, 2016.

[29] C. Potena, R. Khanna, J. Nieto, R. Siegwart, D. Nardi, and A. Pretto, "Agricolmap: Aerial-ground collaborative 3d mapping for precision farming," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1085–1092, 2019.

[30] A. Pretto, S. Aravecchia, W. Burgard, N. Chebrolu, C. Dornhege, T. Falck, F. Fleckenstein, A. Fontenla, M. Imperoli, R. Khanna *et al.*, "Building an aerial-ground robotics system for precision farming: An adaptable solution," *arXiv preprint arXiv:1911.03098*, 2019.

[31] H. Kandath, T. Bera, R. Bardhan, and S. Sundaram, "Autonomous navigation and sensorless obstacle avoidance for ugv with environment information from uav," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2018, pp. 266–269.

[32] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[34] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[35] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[36] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2992–2997.

[37] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.

[38] J. Ku, A. Harakeh, and S. L. Waslander, "In defense of classical image processing: Fast depth completion on the cpu," in *2018 15th Conference on Computer and Robot Vision (CRV)*. IEEE, 2018, pp. 16–22.

[39] M. Burri, H. Oleynikova, , M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *Intelligent Robots and Systems (IROS 2015), 2015 IEEE/RSJ International Conference on*, Sept 2015.

E

frontiers in Robotics and AI

ORIGINAL RESEARCH
published: 19 October 2021
doi: 10.3389/frobt.2021.734548

Coordinating Shared Tasks in Human-Robot Collaboration by Commands

Alexandre Angleraud†, *Amir Mehman Sefat*†, *Metodi Netzev and Roel Pieters**

Cognitive Robotics Group, Faculty of Engineering and Natural Sciences, Tampere University, Tampere, Finland

Human-robot collaboration is gaining more and more interest in industrial settings, as collaborative robots are considered safe and robot actions can be programmed easily by, for example, physical interaction. Despite this, robot programming mostly focuses on automated robot motions and interactive tasks or coordination between human and robot still requires additional developments. For example, the selection of which tasks or actions a robot should do next might not be known beforehand or might change at the last moment. Within a human-robot collaborative setting, the coordination of complex shared tasks, is therefore more suited to a human, where a robot would act upon requested commands.In this work we explore the utilization of commands to coordinate a shared task between a human and a robot, in a shared work space. Based on a known set of higher-level actions (e.g., pick-and-placement, hand-over, kitting) and the commands that trigger them, both a speech-based and graphical command-based interface are developed to investigate its use. While speech-based interaction might be more intuitive for coordination, in industrial settings background sounds and noise might hinder its capabilities. The graphical command-based interface circumvents this, while still demonstrating the capabilities of coordination. The developed architecture follows a knowledge-based approach, where the actions available to the robot are checked at runtime whether they suit the task and the current state of the world. Experimental results on industrially relevant assembly, kitting and hand-over tasks in a laboratory setting demonstrate that graphical command-based and speech-based coordination with high-level commands is effective for collaboration between a human and a robot.

Keywords: collaborative robot (cobot), human-robot interaction, task coordination, knowledge-based, speech recognition

OPEN ACCESS

Edited by:
Matteo Saveriano,
University of Innsbruck, Austria

Reviewed by:
Weiwei Wan,
Osaka University, Japan
Yi-Shiuan Tung,
University of Colorado Boulder, United States

****Correspondence:***
Roel Pieters
roel.pieters@tuni.fi

†These authors have contributed equally to this work and share first authorship

Specialty section:
This article was submitted to Field Robotics, a section of the journal Frontiers in Robotics and AI

Received: *01 July 2021*
Accepted: *01 October 2021*
Published: *19 October 2021*

Citation:
Angleraud A, Mehman Sefat A, Netzev M and Pieters R (2021) Coordinating Shared Tasks in Human-Robot Collaboration by Commands. Front. Robot. AI 8:734548. doi: 10.3389/frobt.2021.734548

1 INTRODUCTION

Collaborative robots (cobots) are at increasing rate being deployed in industrial environments, sharing tasks and the work space with humans (Villani et al., 2018a). Tasks can be individually configured in a human-robot team setting, where the operator demonstrates task sequences and skills for the robot, and the robot repeats them (Ogenyi et al., 2021). This avoids having to go through a development phase, considerably speeding up integration time. Cobots are crucial for this, as they are small, light-weight and can be safely moved around by a human operator (Kumar et al., 2021).

However, this programming of tasks is typically targeted only for independent robot motions, and task execution usually does not include human-robot interaction or physical collaboration. This implies that programming is still done offline, while the robot and the tasks are being prepared, and the actual execution phase is mostly autonomous execution of the robot. While applications can be found (Sadrfaridpour and Wang, 2017; Johannsmeier and Haddadin, 2017; Darvish et al., 2021) that integrate coordinated actions (e.g., waiting for human input or trigger), still this is pre-programmed and planned to happen at certain specified occurrences. Coordination is thus planned in advance and both agents (i.e., human and robot) act as decided by a fixed protocol. If and when problems occur, or when changes need to be made in the collaboration, the work flow is disrupted and has to be restarted when problems get fixed or when changes are implemented. This limitation affects the natural collaboration and fluency between human and robot (Hoffman, 2019), as no spontaneous actions are allowed besides simply halting the robot and the action plan. While exceptions exist (see e.g., (Darvish et al., 2021), which takes into account last-minute changes of task allocation), task plans are typically short, to avoid a large task plan network that is complex to model and track.

To allow more natural and fluent human-robot interaction, we believe collaboration between human and robot should be coordinated by the human, assisted by the robot and its knowledge and reasoning capabilities. At any given time during the collaboration, the human worker should be able to select suitable actions from the robot to assist the shared task. The robot verifies that the action is suitable and possible, based on its current state of the world and capabilities. Such knowledge is incorporated in a knowledge base that is updated at regular intervals by observations and human instructions. The selection of actions for the robot thus requires human commands to allow for intuitive instructions. Speech and text-based commands are most suitable as, similar to human-human communication (Rocci and Saussure, 2016), semantics can be included.

In this work, we present the developments to allow human coordination in shared human-robot collaborative tasks. The main contributions of this paper are:

- A knowledge-based system architecture that supports reasoning, planning and knowledge integration
- Shared task coordination by human commands, either by a graphical interface or by speech
- Industrially relevant use case scenarios that evaluate the approach

The paper is organized as follows. **Section 2** reviews the state of the art in human-robot collaboration and verbal communication in robotics. **Section 3** presents the proposed system, with the knowledge and reasoning architecture (**Section 3.2**) that describes the state of the world, the actors present within it and the capabilities and properties each contain. Then, in **Section 3.3** the selection of robot actions is enabled by both a graphical command-based and speech-based user interface that is connected to the knowledge base for reasoning over capabilities and actions. Results of the approach are presented in **Section 4** by evaluation of human-robot collaborative tasks inspired from real industrial use cases. **Section 5** presents a discussion on the work, including its limitations. Finally, **Section 6** concludes the work.

2 RELATED WORK

2.1 Human-Robot Collaboration

Collaboration between human and robot within industrial environments has received considerable attention in recent years (Villani et al., 2018a; Kumar et al., 2021). Clear distinctions are made between different categories of collaboration, for example, whether tasks and the environment are shared and which agent takes which task (Kolbeinsson et al., 2019). This allocation of tasks requires careful planning and depends on several (in)dependent factors, such as the capabilities of the robot, the difficulty of re-programming and re-configuring the setup, complexity of the task, among many others. Cobots are well suited to be integrated in such environments, due to their light weight, integrated safety functions and human-centered robot programming interfaces (Villani et al., 2018a,b). Industrial integration requires adherence to international standards that assess the safety aspects (i.e., (ISO-10218-1/2:2011, 2011), for industrial robots and systems, and (ISO-15066:2016, 2016), for collaboration) by a formal risk assessment, where, besides the robot itself, additional systems (Halme et al., 2018) can be incorporated to guarantee safety of the human worker. Additional trends in collaboration between human and robot take the fluency of interaction (Hoffman, 2019) or human factors (Chen and Barnes, 2014) into account. This implies that the user experience (Chowdhury et al., 2020) and user acceptance (Müller-Abdelrazeq et al., 2019) is considered by design of the interaction, with suitable technology that improves, instead of hinders, the outcome.

Even though much research and development is ongoing to accelerate the uptake and deployment of collaborative robots, there is no universal solution that fits all. This is perhaps best exemplified by the variety of modalities available for interaction and the magnitude of differences in industrial environments, tasks and contexts. Several different modalities have been utilized for communication, as demonstrated for gestures (Liu and Wang, 2018), augmented and virtual reality (Dianatfar et al., 2021), verbal and non-verbal communication (Mavridis, 2015) and physical interaction (Ogenyi et al., 2021).

The mentioned works on human-robot collaboration demonstrate that communication is crucial in achieving the goals of the interaction. Depending on the modality, this information exchange can take many forms, such is robot goal poses, safety zones, basic commands, task messages, etc. Non-verbal commands, however, typically transmit different information, as compared to verbal commands. Human to human communication, for example, thrives in verbal communication (Rocci and Saussure, 2016), as information can be shared efficiently and with different nuance and meaning. Enriching robots with the capabilities to interpret, understand and react to verbal commands, or even natural language, is, however, still in early stages of development.

2.2 Verbal Communication in Robotics

Verbal interaction between humans and robots has seen success in many different cases (Mavridis, 2015; Marin Vargas et al., 2021). Often, literal commands provide the robustness for communication, as the commands are known, and only basic, short sentences are utilized. The step of going beyond literal command-based instructions aims at extending communication to include semantic annotations of commands (Dukes, 2013) or purely natural language (Williams et al., 2015). One advantage of natural language, as compared to literal commands, is the inclusion of semantics, enabling similar expressions in different ways, such that it is most convenient and comfortable for the human. Moreover, higher-level (cognitive) concepts, such as intention, emotion and action, can be (in) directly included in a phrase, as typically present in everyday human language. The extraction of such information for a Natural Language Processing (NLP) system is, however, not an easy feat. State of the art approaches, utilizing deep neural networks (Otter et al., 2022) or other learning based techniques (Sharma and Kaushik, 2017), have shown real-time conversational skills, as, for example by IBM's Watson (High, 2012) or GPT-3 (Brown et al., 2020).

With respect to robotics, the understanding and acquisition of language can take advantage of the situational nature of a robot, as it is placed in a dedicated environment where tasks and context are known (Taniguchi et al., 2019). Research works have focused on specific contexts for extractions and interpretations of robot instructions, such as manipulation (Misra et al., 2016), grasping (Chen et al., 2021), intention recognition (Mi et al., 2020; Sun et al., 2021) and grounding (Misra et al., 2016; Shridhar et al., 2020; Vanzo et al., 2020). Other approaches interpret natural language through human-robot dialog (Thomason et al., 2015), or utilize additional sensor modalities, such as vision (Sun et al., 2021; Chen et al., 2021). Research has also targeted semantics, both to understand the world and to execute robot actions within it (Ramirez-Amaro et al., 2019). Approaches specific to learning or assigning the semantics of assembly tasks can be found in (Stenmark and Malec, 2014; Savarimuthu et al., 2017).

Most of the presented works consider the tasks as fixed, with little variation in task allocation (Johannsmeier and Haddadin, 2017) or with a low number of total tasks to be executed (Darvish et al., 2021). The reason for this is that with increasing variation in tasks, the task models easily become too large to manage and track. However, when considering Industry 4.0, the trend of smart manufacturing pushes production processes to include wide variations in products, which are to be completed at irregular and unknown time instances. Collaboration between human and robot is suitable to achieve this with higher efficiency than robots (i.e., full automation) or humans (i.e., full manual labour) alone, as it avoids large and complex task plans that include all possible product variations, and avoids large robot programming efforts. The problem then becomes how to command and coordinate robots effectively and efficiently.

In this work, we address the collaboration between human and robot from the point of view of coordination. In order to enable fluent collaboration, human coordination decides when and which robot actions should be executed. This is done by human command phrases (actions and targets), that can be communicated by speech or via a graphical user interface, at any given time during the shared task. Reasoning over the knowledge base that holds an up-to-date world model, then ensures that robot tasks are executed at the correct time (e.g., when the robot is free) and with the correct functionalities (e.g., robot is capable to reach an object). Command phrases, in combination with a dedicated knowledge representation of the world, has the advantage of including semantic annotation to all knowledge, making the system customizable to the user (e.g., by preferred phrases) and to the tasks (e.g., no predefined task plan, but the user decides who does what and when).

3 MATERIALS AND METHODS

The methodology of the proposed approach and its materials are explained by the system architecture and its contents, which includes the knowledge base and reasoning, action planning and the different interaction modalities, i.e., graphical command-based and speech-based.

3.1 Terminology

The terminology, used throughout the work is clarified as follows:

Coordination-the act of managing actions towards a common goal, while handling problems, conflicts and collaboration.

Communication-the exchange of information by different modalities.

Command-a word that the robot knows and reacts to.

Natural Language Processing (NLP)-refers to the computational approach of analyzing, understanding and manipulating natural language text or speech.

Automated Speech Recognition (ASR)-converts spoken language to text.

Semantic annotation-is a process of attaching relevant (meta) data.

In context of the human-robot collaborative tasks, our contributions lie in the human coordination of robot actions by utilizing commands, which are known in the system. Automated Speech Recognition (ASR) tools capture the spoken commands and convert them to text. Natural Language Processing (NLP) takes the text and matches them to existing or related phrases by semantic information that is annotated to the contents of the knowledge base. To reduce the complexity in modelling and tracking shared task plans, the tasks commanded to the robot are short (only few actions) and are not integrated in a higher level goal. This implies that a (shared) goal is only taken into account by the human, who coordinates the actions of him/herself and the robot. Nevertheless, as the world and the low-level tasks are represented in an ontology, this allows for their evaluation, before a robot action is executed. Practically, the current state of the world and the task, and the requested commands are evaluated for matching conditions and capabilities. For example, whether the robot can reach a destination or is holding an object for placement.

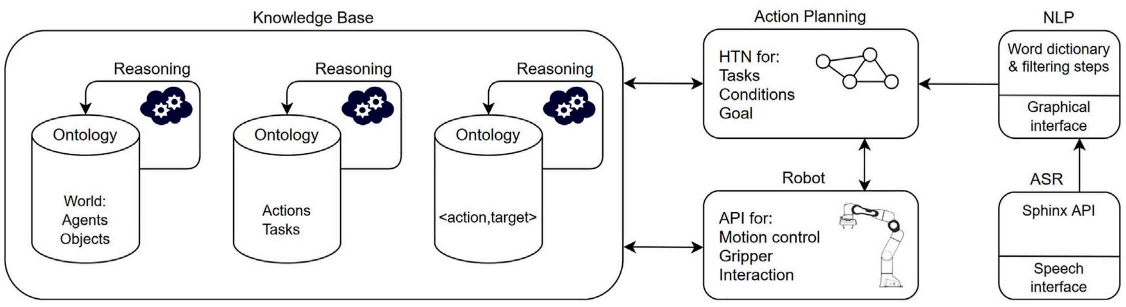


FIGURE 1 | The system architecture is divided in several blocks as follows. The knowledge base (KB) holds all knowledge of the world in form of ontologies, which are updated by reasoning and human input. Action planning generates shared plans in form of hierarchical task networks (HTN), which take input from the KB. Robot capabilities are integrated via a separate robot application programming interface (API) where skills and motion primitives are defined. Automated speech recognition (ASR) and natural language processing (NLP) are separate modules that provide the human input for coordinating the collaborative tasks.

TABLE 1 | Coordination of shared tasks is commanded by `<action,target>`-pairs that specify an action to be executed by the robot, with an accompanying target. Specific details of the architecture are as follows: Pre-conditions-checks whether certain conditions of the world and its content prior to execution are met (e.g., object location, state/capabilities of the robot). Signature-specifies onto which the action/target acts; object, robot and/or human. Semantics-lists the different commands that can be used for triggering the same action/target. Format-describes the underlying knowledge format. Explanation-provides details of the action/target and its specific (sub) tasks.

Action	Pre-conditions	Signature	Semantics	Format	Explanation
moveTo	isWithinReach isReady	Object	Come Go	move action	Move robot end-effector
graspObject	gripperEmpty isReady holdsObject	Object Robot	Pick Take	motion action gripper action	Grasps object
placeObject	isWithinReach isReady	Object Robot	Place Deposit	motion action gripper action	Places object
handOver	isWithinReach isReady humanPresent	Object Robot Human	Give Hand	motion action gripper action	Hand-over object
kitParts	isWithinReach isReady	Object Robot	Kit Stock	motion action gripper action	Pick and place objects
Target					
Parts	isWithinReach canBeGrasped	Object Robot Human	Bolt Bolts Tool	3D Pose	Location of parts
Box	isWithinReach isReady isEmpty	Object Robot	Box Kit Container	3D Pose	Location of box
Table	isWithinReach isReady isEmpty	Object Robot	Storage Kit_store Back	3D Pose	Pose on table
Human	isWithinReach isReady humanPresent	Object Human	Here Me	3D Pose	Human hand-over pose

3.2 System Architecture

The architecture of our system is based on previous developments on knowledge-based planning for human-robot collaborative tasks (Angleraud et al., 2018). One crucial difference is that no high-level planner is utilized and autonomous robot actions, as inferred from the knowledge base and reasoning, are excluded. Instead, human coordination decides which actions are selected and executed, verified by the reasoning module. The system is depicted in **Figure 1** and the individual modules are explained in detail, as follows.

3.2.1 Knowledge Base and Reasoning

Knowledge on the world and its content is represented by ontologies, and referred to as the Knowledge base (KB). As main advantage, ontologies offer a structured description of knowledge, its domain and the relationships that hold between its contents. The KB contains the objects and agents present, and includes relevant information for the tasks and the goals, such as their location, pose, status, etc. Executable actions of the robot, such as end-effector motion, grasping and object placement are expressed as an $<$ `action,target` $>$ -pair that can be called by the human at any requested instance. The KB and its knowledge representation allow for relationships to be defined between actions, targets and the world state, such that dependencies and conditions can be checked in order to update the KB. Moreover, relationships enable verification of conditions for robot action execution. Reasoning over the KB, therefore, serves two functions:

World update - Observations external and internal from robot and the world are utilized to update the KB. For example, the state of the robot, such as end-effector pose, gripper state, and its actions being executed. Moreover, human commands (i.e., $<$ `action,target` $>$ -pairs) are used to update the KB.

Action execution checks-Relationships between entries of the KB are checked when robot actions are queried. These pre-conditions verify and enable the execution of robot actions.

Table 1 lists a subset of $<$ `action,target` $>$ -pairs present in the KB, which is utilized for updating the KB and for coordination of the shared human-robot collaborative tasks.

3.2.2 Action Planning

Robot action plans are constructed from mid-level action sequences that execute requested tasks. At a higher level, human coordination guides the collaboration, in order to achieve a shared goal. Action plans are represented by

Hierarchical Task Networks (Georgievski and Aiello, 2015), which take input from the KB to generate a plan. On a practical level, this implies that at each action plan node, the state of the robot and the human is checked (e.g., whether the human is active or not, represented by the is Ready state). When the robot is ready, additional pre-conditions of the world are verified that assess whether the actions can be executed (e.g., end-effector pose can be reached: is withinReach, object present: can Be Grasped, gripper empty: gripper Empty; see **Table 1**, pre-conditions column). When verified correct, the actions are executed. One example explains this planning concept. A robot grasping task is planned as a sequence of actions (i.e., robot motion, gripper motion), where each node in the plan represents the different steps in between the robot actions. At each node, pre-conditions are checked, towards the state of the world and the actions requested, prior to execution. A high level understanding of the shared task is therefore not present in the KB, but only the actions that can be requested from the human. This simplifies the formal planning definition and leaves the high-level coordination towards the shared goal to the human.

3.2.3 Semantic Annotations

Ontologies are well suited to incorporate (semantic) annotations to knowledge. Properties, relations and dependencies can be easily connected to individual entities and link entities together to form (chains of) relationships. **Table 1** lists few examples in the pre-conditions column that represent relations and attributes in the world. In regards to the interpretation of semantics towards robot commanding, our system offers the incorporation of semantic annotations, as selected commands can be assigned to address specific actions and targets (see Semantics column in **Table 1**). Integration of such additional semantics requires the requested commands to be included in the NLP dictionary and the planning domain ontology.

3.3 Interaction Modalities

Interaction between human and robot can be divided into modalities utilized for programming robot actions and modalities utilized for task coordination. Physical interaction, such as hand-guiding a robot motion, demonstrates an end-effector pose and is part of a set of robot capabilities developed by the robot manufacturer (i.e., gravity-compensated hand-guiding). Here, we focus on the core interaction modalities of our work, i.e., a graphical command-based interface and a speech-based interface.

3.3.1 Graphical Command Interface

The graphical command-based interface enables a human to instruct actions to the robot, by an $<\texttt{action,target}>$-pair selected from a graphical user interface (GUI). Based on the current state of the world, a single action can be selected, followed by a suitable target (see **Table 1**). It has to be noted that the semantics of the actions and the targets are not fixed and can be arbitrarily chosen by the human, by simply changing the terminology in the specific ontology.

3.3.2 Speech Interface

An ASR module enables the shared human-robot collaborative tasks to be coordinated by verbal commands. This essentially relies on the same functionality as the command interface but now, speech has to be interpreted and connected to individual actions and targets to form $<\texttt{action,target}>$-pairs. While ASR depends on an external software tool, several NLP steps and filters are included to our proposed system. To reduce the complexity of the NLP steps, several additional requirements are set for the acceptance of the command phrases. These are explained as follows.

Word exclusion-Common words are removed from a phrase, such as articles (e.g., 'the', 'a', 'an')

Word limit-A maximum of two words are accepted for processing.

As general rule, from a command phrase, the NLP system accepts only the words that are defined in the dictionary, and a command phrase should only contain one action and one target. In other cases (e.g., multiple actions/targets or one action/target missing), the command is not accepted. Following, it is checked whether there exist properties of the action and the target, such that a meaningful task can be extracted for the robot. This is done by reasoning and verification over the knowledge base, where all actions and targets are described by suitable properties and relationships.

3.3.3 Knowledge Integration

Integrating new knowledge into the system can be done in various ways, depending on the type of knowledge and its format, as summarized in **Table 2**. Robot actions to be included are divided in primitive actions, such as single motions or gripper actions, and tasks, which are a list of actions. In both cases the action is demonstrated by the human or programmed in the action library of the robot and linked to the ontology by suitable function call definition. As part of the ontology, conditions should then be defined that will be evaluated before action execution.

Targets, which can be locations in 3D space and target objects to grasp (see **Table 1**), are defined as 3D poses in the world space, to be utilized for the robot, and defined by either hand-guiding the robot or by hard-coding the pose into the KB. This means objects and robot motion are not predefined, but are taught to the system before the execution of the shared task. Annotations to the target in the ontology can be included to provide additional and sufficient information to the target pose. For example, objects such as boxes, into which objects can be placed, require a pose that denotes the location within the box, instead of the pose of the box itself.

Direct inclusion of alternative words (synonyms) for existing commands in the ontologies can be easily done via an ontology editor, such as Protégé (Musen, 2015). This enables semantics to be added to all knowledge, by taking advantage of the functionalities of the OWL2 language. Similarly, new reasoning rules can be added by defining new rules in the SWRL language[1].

4 EXPERIMENTS

In this section we present the results of our work, by describing and evaluating two use case scenarios that are representative for industrial human-robot collaboration.

[1]https://www.w3.org/Submission/SWRL

TABLE 2 | Procedure for integrating new knowledge into the system.

Action	Format	Modality	Explanation
Primitive	Robot action	Software integration Python and ontology	Primitive robot actions can be included by function call from ontology to action library
Task	List of robot actions	Software integration Python and ontology	Higher level tasks can be included by defining a list of robot actions
Target			
Pose/object	3D pose	Robot hand-guiding	New targets are defined by hand-guiding the robot to a desired pose. This target is then recorded in the ontology
Other			
Reasoning rule	SWRL	Software integration Python and ontology	New reasoning rules are defined in the SWRL language and integrated to update the ontology
Synonym	Words	Ontology population	Synonyms to all actions and targets can be included by creating new ontology instances

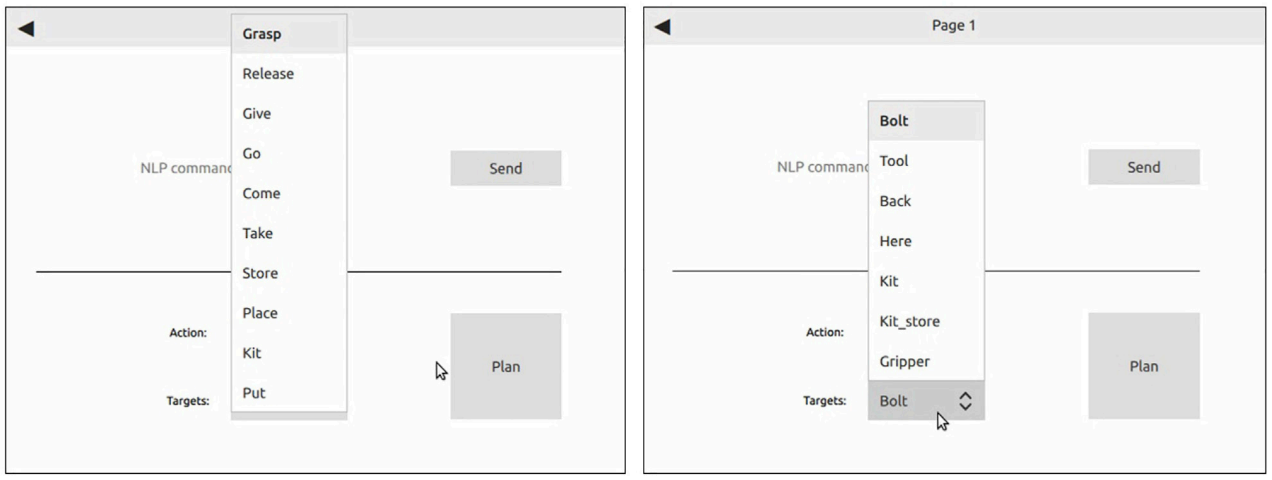


FIGURE 2 | The graphical command interface enables actions and targets to be selected from a drop-down menu. Left: available actions. Right: available targets.

4.1 Implementation

The system architecture and interaction modalities are developed in Python3, utilizing ROS for robot communication and control. The graphical user interface (see **Figure 2**) is developed in Qt and is launched as single interaction mechanism, enabling also the speech module to pass commands to the system. The Google Speech Recognition engine[2] enables spoken words to be converted to text. Ontologies are defined using OWL2 standard[3] with owlready2[4] and Protégé (Musen, 2015) as ontology editor. Reasoning over the knowledge is done by evaluating rules in the SWRL language1.

Use case scenarios are demonstrated with the Franka Emika Panda[5] collaborative robot that provides robust motion profiles and control actions for object pick-and-place and hand-over tasks. Industrial parts and tools from a local Diesel engine manufacturer are utilized to demonstrate the capabilities of the proposed system, which includes (collaborative) tasks for the assembly of Diesel engine components and human-robot hand-over tasks for robot assistance. The work environment consists of two tables; one for the robot and parts/tools to be placed, and one for the human operator and the Diesel engine assembly. Both tables are accessible for the human operator and the robot, implying that the whole environment is a shared work space. For the robot, two separate supportive tasks, and thus experiments, are defined:

1. **human-robot hand-overs of parts/tools**-enables the robot to act as a support to workers, while they are engaged in a Diesel engine (dis)assembly task. Hand-over actions from robot to human and from human to robot are coordinated with industrial parts, such as bolts and tools. Human actions include assembly operations of parts to a Diesel engine and the handling of tools.
2. **Robot assisted kitting**-enables the robot to group individual items into relevant kits. This assists the human operator in a Diesel engine disassembly procedure, for example by collecting and keeping track of all parts. As a separate activity, kitting occurs alongside the human disassembly procedure. Parts to be handled are bolts and hand-tools.

[2]https://pypi.org/project/SpeechRecognition
[3]https://www.w3.org/TR/owl2-syntax
[4]https://pypi.org/project/Owlready2
[5]https://franka.de

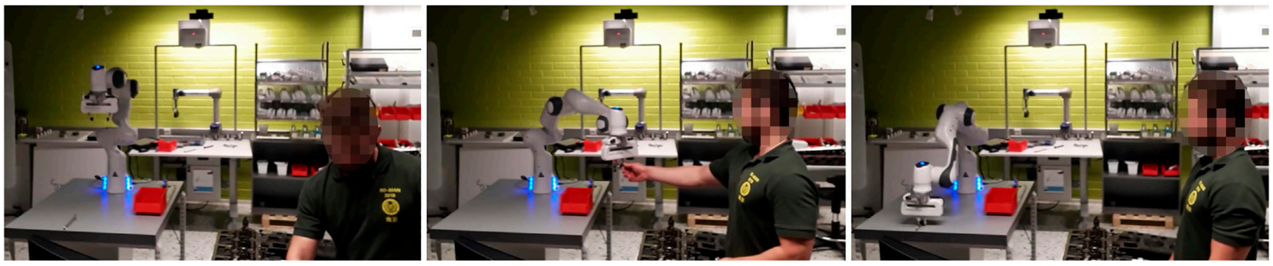
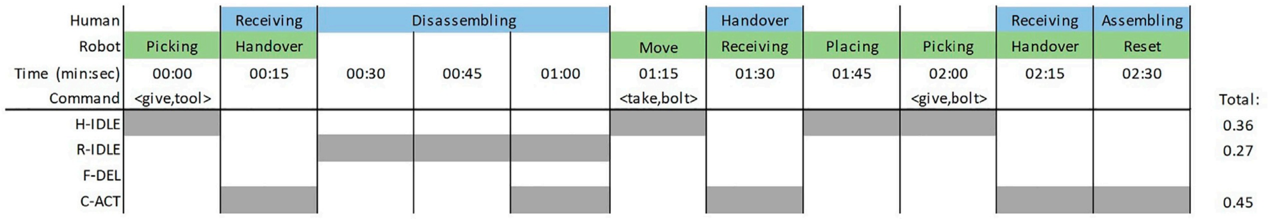


FIGURE 3 | Different stages of the hand-over scenario by graphical command-based coordination. Left: 00:00 - command `<give,tool>`is send, which instructs the robot to hand-over the wrench from the table to the person. Middle: 01:30 - commands `<come,here>`and `<take,bolt>`are used to instruct the robot to receive a bolt from the person. Right: 01:45 - command `<give,bolt>`is used to instruct the robot to pick up and hand-over a bolt from the table to the person.



Human		Receiving	Disassembling				Handover			Receiving	Assembling	
Robot	Picking	Handover				Move	Receiving	Placing	Picking	Handover	Reset	
Time (min:sec)	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	02:00	02:15	02:30	
Command	<give,tool>					<take,bolt>			<give,bolt>			Total:
H-IDLE	▨					▨		▨	▨			0.36
R-IDLE			▨	▨	▨							0.27
F-DEL												
C-ACT		▨			▨		▨			▨	▨	0.45

FIGURE 4 | Task assignment chart for hand-over tasks by graphical command-based coordination. The chart depicts the actions of the human (blue) and the robot (green) in the shared collaborative scenario. The fluency metrics indicate a relatively low human and robot idle time (H-IDL and R-IDL) and high concurrent activity (C-ACT). Functional delay (F-DEL) is avoided completely.

Both tasks are evaluated by the graphical and speech command interface. A video of the experiments can be seen here: https://youtu.be/SzIuLHzLYpA. In addition, the hand-over task is compared to two baseline methods (i.e., strict turn taking and fast robot cycle), from which clear objective metrics can be extracted that assess the fluency of coordination (Hoffman, 2019). The metrics are human idle time (H-IDL), robot idle time (R-IDL), functional delay (F-DEL) and concurrent activity (C-ACT). For all experiments metrics were calculated with a resolution of 15 sec. This was chosen to be coherent with all experiments and their analysis. In practice, these results were obtained from analyzing the videos of the experiments and finding a common resolution between the different robot and human actions. Experiments were repeated five times. The experimental scenarios are as follows.

4.2 Use Case Scenario 1: Command-Based Collaboration

To evaluate human-robot collaboration by commands, a scenario is defined where a human operator selects robot actions (in form of $<$ `action,target` $>$ -pairs) from a graphical user interface (GUI). The list of actions and targets can be selected from a drop-down menu in the GUI, as depicted in **Figure 2**. Both tasks, i.e., human-robot hand-over and kitting, are demonstrated as follows.

Figure 3 depicts different stages of the human-robot hand-over scenario by graphical command-based coordination. **Figure 4** depicts a task assignment chart, which visualizes when different agents, i.e., robot or human, are active and with what activity. The commands requested by the human and utilized for robot coordination are depicted as well, and demonstrate the variation in robot actions and how they can be requested. In this case, commands are utilized for object picking and placing (i.e., `give` and `take`), robot motion (`come`) and hand-over tasks from robot to human and human to robot. Parts and locations are described by `tool`, `bolt` and `here`.

Figure 5 depicts different stages of the kitting scenario by graphical command-based coordination. **Figure 6** depicts a task assignment chart, which visualizes when the robot is active and with what activity. The commands requested by the human and utilized for robot coordination are depicted as well, and demonstrate the variation in robot actions and how they can be requested. In this case, commands are utilized for object picking and placing (`pick`, `place`, `kit` and `take`) and robot motion (`go`). Parts and locations are described by `tool`, `box`, `back`, `bolts`, `kit` and `kit_store`.

4.3 Use Case Scenario 2: Speech-Based Collaboration

To evaluate human-robot collaboration by speech, a scenario is defined where a human operator requests robot actions (in form of $<$ `action,target` $>$ -pairs) by speech. A list of actions and targets are available, which are known by the human operator. Again, both tasks, i.e., human-robot hand-over and kitting, are demonstrated as follows.

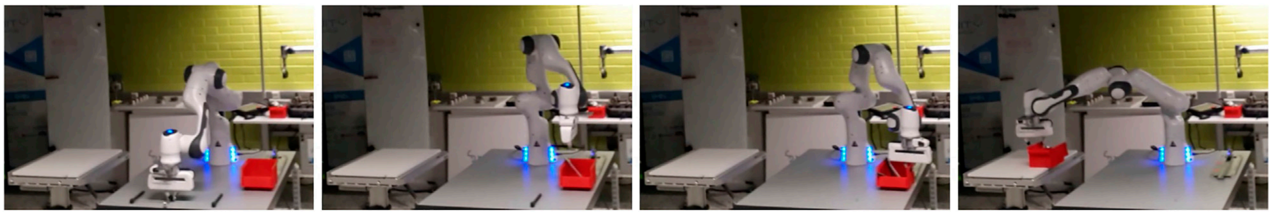
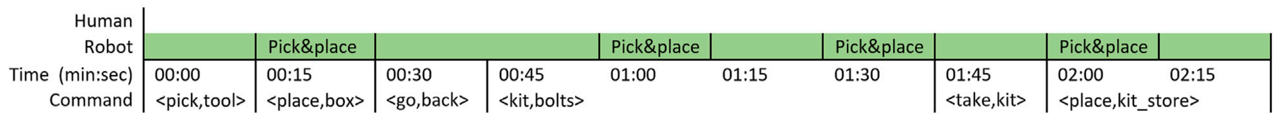


FIGURE 5 | Different stages of the kitting scenario by graphical command-based coordination. Left: 00:00 - commands `<pick,tool>`and `<place,box>`are send, which instructs the robot to pick and place the tool from the table to the kit. Second left: 01:00 - command `<kit,bolts>`is used to instruct the robot to place all bolts from the table into the kit. Second right: 02:00 - command `<take,kit>`is used to instruct the robot to pick up the kit from the table. Right: 02:15 - command `<place,kit_store>`is used to instruct the robot to place the kit on the other table.



Human										
Robot	🟩	Pick&place	🟩	🟩	Pick&place	🟩	Pick&place	🟩	Pick&place	🟩
Time (min:sec)	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	02:00	02:15
Command	<pick,tool>	<place,box>	<go,back>	<kit,bolts>				<take,kit>	<place,kit_store>	

FIGURE 6 | Task assignment chart for the kitting task by graphical command-based coordination. The chart depicts only the actions of the robot (green), as it acts alone.

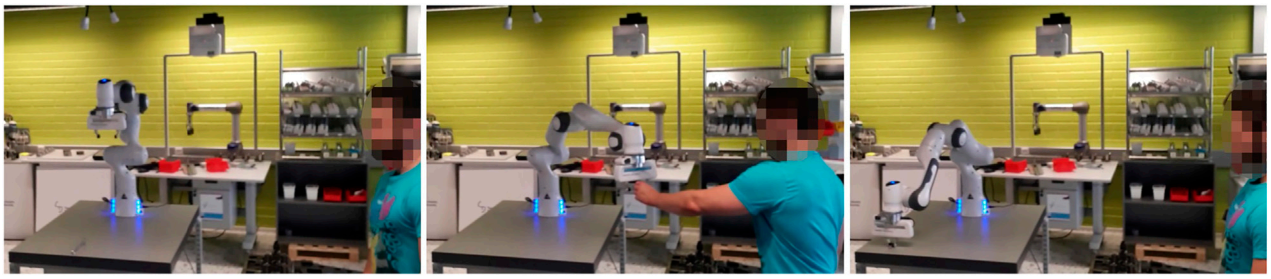
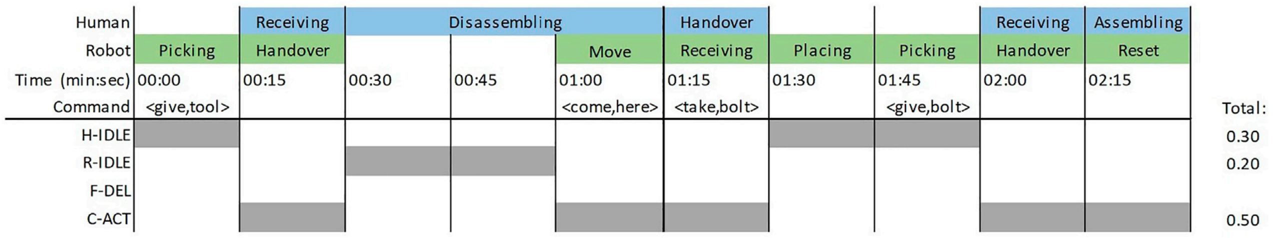


FIGURE 7 | Different stages of the hand-over scenario by speech command-based coordination. Left: 00:00 - command `<give,tool>`is spoken, which instructs the robot to hand-over the wrench from the table to the person. Middle: 01:30 - commands `<come,here>`and `<take,bolt>`are spoken to instruct the robot to receive a bolt from the person. Right: 01:45 - command `<give,bolt>`is spoken to instruct the robot to pick up and hand-over a bolt from the table to the person.



Human		Receiving	Disassembling			Handover			Receiving	Assembling	
Robot	Picking	Handover			Move	Receiving	Placing	Picking	Handover	Reset	
Time (min:sec)	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	02:00	02:15	
Command	<give,tool>				<come,here>	<take,bolt>		<give,bolt>			Total:
H-IDLE	▨						▨	▨			0.30
R-IDLE			▨	▨							0.20
F-DEL											
C-ACT		▨			▨	▨			▨	▨	0.50

FIGURE 8 | Task assignment chart for hand-over tasks by speech command-based coordination. The chart depicts the actions of the human (blue) and the robot (green) in the shared collaborative scenario. The commands requested by the human to the robot are identical to the graphical command-based scenario. The fluency metrics indicate a relatively low human and robot idle time (H-IDL and R-IDL) and high concurrent activity (C-ACT). Functional delay (F-DEL) is avoided completely.

Figure 7 depicts different stages of the human-robot hand-over scenario by speech-based coordination. **Figure 8** depicts a task assignment chart, which visualizes when different agents, i.e., robot or human, are active and with what activity. The commands requested by the human and utilized for robot coordination are depicted as well, and demonstrate the variation in robot actions and how they can be requested. In this case, commands are identical to the graphical command-

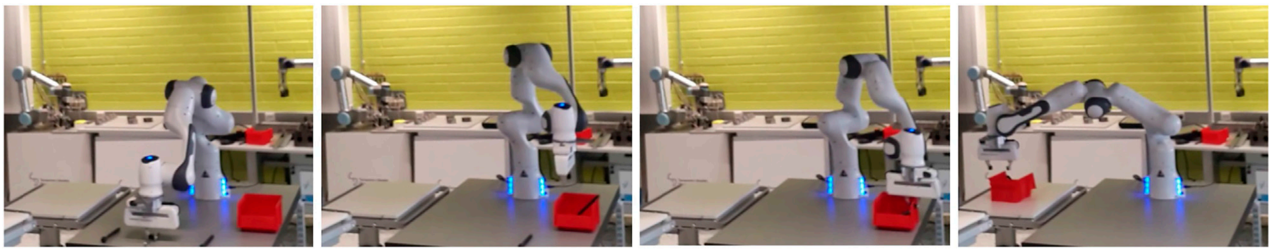


FIGURE 9 | Different stages of the kitting scenario by speech command-based coordination. Left: 00:00 - commands `<take,tool>`and `<deposit,container>`are spoken, which instruct the robot to pick and place the tool from the table to the container. Second left: 01:00 - command `<stock,bolts>`is spoken to instruct the robot to place all bolts from the table into the container. Second right: 02:00 - command `<take,container>`is spoken to instruct the robot to pick up the kit from the table. Right: 02:15 - command `<deposit,storage>`is spoken to instruct the robot to place the kit on the other table.

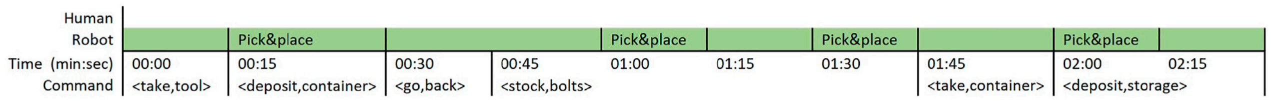


FIGURE 10 | Task assignment chart for kitting by speech coordination. The chart depicts only the actions of the robot (green), as it acts alone. The commands requested by the human to the robot are different from the graphical command-based scenario.

based hand-over scenario, i.e., object picking and placing (i.e., `give` and `take`), robot motion (`come`) and hand-over tasks from robot to human and human to robot. Parts and locations are again described by `tool`, `bolt` and `here`.

Figure 9 depicts different stages of the human-robot hand-over scenario by speech-based coordination. **Figure 10** depicts a task assignment chart, which visualizes when different agents, i.e., robot or human, are active and with what activity. The commands requested by the human and utilized for robot coordination are depicted as well, and demonstrate the variation in robot actions and how they can be requested. In this case, commands are utilized for object picking and placing (`take`, `deposit` and `stock`) and robot motion (`go`). Parts and locations are described by `tool`, `container`, `back`, `bolts`, `container` and `storage`.

In all cases, robot actions are requested at an instance as decided by the human operator and are executed without major delay, if the robot is not active in other tasks. If the robot is busy, the action is executed as soon as the robot becomes available again.

4.4 Baseline Comparison

In order to assess the fluency of collaboration between the human and the robot, resulting from our coordination approach, we devised two baseline approaches as comparison:

1. **Strict turn taking**-each action is immediately followed by the next action of the other teammate, implying that tasks are done sequentially, instead of parallel. Exceptions are the hand-over actions as they require both agents to complete.
2. **Fast robot cycle**-each robot action is executed as fast and as soon as possible, irrespective of the actions of the human.

In both baseline approaches the actions of the robot are not commanded by a human, but executed according to a predefined protocol, as could be found in a factory automation setting. From the objective fluency metrics the following conclusions can be drawn. In strict turn taking (see **Figure 11**) the human idle time (H-IDL) is relatively high (0.46 or almost half of the time), as concurrent activity (C-ACT) is avoided. Exceptions are the handover tasks that require both agents to collaborate. Similarly, the robot idle time (R-IDL) is also relatively high (0.31 or almost a third of the time), due to the similar reason. The functional delay (F-DEL), however, is avoided, as the agents are never waiting for the completion of each other's action and are never idle at the same time. In the fast robot cycle approach (see **Figure 12**) the idling time of both the human (H-IDL, 0.30 or a almost a third of the time) and the robot (R-IDL, 0.20 or one fifth of the time) are low, indicating an efficient utilization of resources. Moreover, the concurrent activity (C-ACT) is high (0.5 or half of the time). The functional delay (F-DEL) is, again, avoided. Following, we compare the baseline approaches to the proposed coordination approaches.

5 DISCUSSION

Based on the experiments presented in **Section 4**, here we discuss and compare their outcome, and present limitations and future work.

5.1 Comparison to the Baselines

Human-robot collaboration fluency can be compared in detail according to the metrics of human and robot idle time (H-IDL and R-IDL), functional delay (F-DEL) and concurrent activity

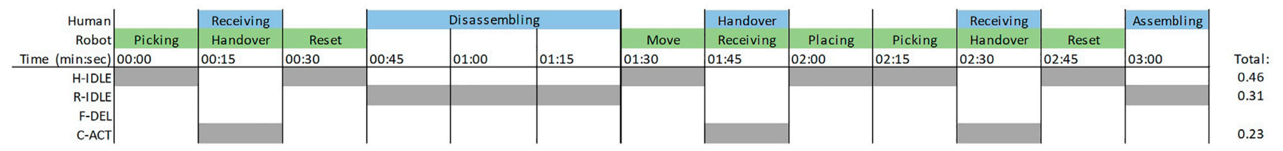


FIGURE 11 | Task assignment chart for the baseline approach of strict turn taking, where each action is immediately followed by the next action of the other teammate. The fluency metrics indicate a relatively high human and robot idle time (H-IDL and R-IDL) and low concurrent activity (C-ACT). Functional delay (F-DEL) is avoided completely.

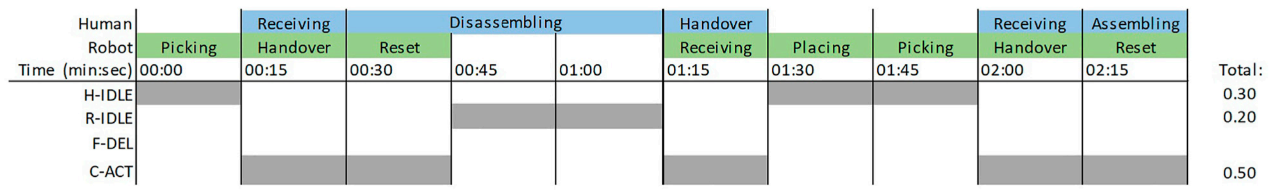


FIGURE 12 | Task assignment chart for the baseline approach of fast robot cycle, where each robot action is executed as fast and as soon as possible. The fluency metrics indicate a relatively low human and robot idle time (H-IDL and R-IDL) and high concurrent activity (C-ACT). Functional delay (F-DEL) is avoided completely.

(C-ACT). In all cases functional delay is avoided, as the agents are never waiting for the completion of each other's action and are never idle at the same time. In both command-based approaches the fluency metrics are roughly the same, as the commands are requested at similar time instances during the collaborative task. This indicates a relatively high rate of concurrent activity (C-ACT, almost half of the time) and a relatively low human and robot idle time (for all cases almost a third of the time or less). Compared to all other approaches (fast robot cycle baseline and both command-based approaches) the strict turn taking baseline has the worst performance, with higher idling times (H-IDL and R-IDL), a lower concurrent activity (C-ACT) and the longest scenario execution time. The baseline approach of fast robot cycle has a very close performance compared to the command-based approaches, with minor differences in when actions are executed. This indicates that the proposed command-based approaches are very efficient as the robot has a high utilization rate. However, the most important benefit, which cannot be measured by the fluency metrics, is not possible for both baseline approaches. That is, the flexibility to coordinate and command the actions of the robot at any time and any rate.

5.2 Coordination by Commanding

Coordination by commands gives control to the human operator to direct at his/her level of interaction and pace. This implies that its not determined beforehand which tasks are shared and in what level of interaction, leading to an inherently flexible system that suits a wide variety of collaboration. This level of flexibility is not present in the baseline approaches, which assume a predefined sequence of actions, at a fixed pace. Both coordination scenarios demonstrate fluent collaboration between human and robot that is not predefined by a fixed task sequence. High-level robot tasks that contribute to the shared goal (assembly) are object pick and placement and physical interaction for human-robot hand-overs (haptic cues). During each shared task (2.5 minutes), multiple robot commands are requested, i.e., pick and place, and hand-over actions, all while the human operator is engaged, and not disturbed, in the (dis)assembly procedure.

However, a graphical user interface (GUI), even if it approaches the capabilities of a speech recognition system, can be unsuitable for industrial environments. The main reasons identified for this are as follows. First, a GUI takes attention away from the task and the shared environment. Even though this does not necessarily imply danger, it could halt the work or even lead to a reduction in work quality and efficiency. Second, industrial tasks, such as assembly, often require manual handling or manipulation, which cannot be interrupted at random. Collaborative actions would need to be halted and parts would need to be put down in order to interact with the GUI.

Despite these limitations, supportive functions to the GUI can be included to enhance and simplify the interaction. The selection of tasks can be narrowed down by reasoning assistance on the current state of the world (what robot actions are possible) and the actions commanded by the human (what actions are most likely to be needed). This means that only the actions that are suitable at the current moment are available to command and other actions are removed from the selection list.

5.3 Commands Vs Speech

To humans, speech is one of the dominant modalities for direct communication (Rocci and Saussure, 2016). In a collaborative work scenario, where manual tasks are taking most attention, speech can be utilized for directing actions and queries to co-workers and, as demonstrated in this work, to robots. However, in industrial environments background noise is very likely to interfere with the reliable recognition of speech. The graphical command interface is one alternative that can replace the recognition of speech, while still ensuring the same functionality of the system. Unfortunately, and unsurprisingly, the graphical interface is less convenient than speech, as it requires manual operation and takes attention away from the task at hand. As a result, robot commanding by GUI takes more time leading to less efficient operations. A disadvantage of the speech interface is it reliability in recognizing correct speech commands and connecting them to the correct action or target phrase. Speech can be misinterpreted and strict guidelines need to be in place that specify how phrases are verbalized. This issue is not present in the GUI interface, as only existing < `action,target` > -pairs can be selected.

One matter that holds for both interface modalities, is the number and format of commands. As robot skills are plentiful, a limit should be set to how many commands (actions and targets) are available to be executed. In practice, the number of commands to be memorized by the human operator should be limited, as looking up commands from a cheat-sheet has negative effects to a desired fluent collaboration. Likewise, scrolling through a long list of commands from a GUI has the same negative effect. Commands should be intuitive, such that they are directly understandable by the human operator, thereby representing their functionality.

5.4 Limitations and Future Work

As demonstrated by the use cases, the system is currently limited by the low number of robot actions and their complexity. However, additional actions, such as motion trajectories, advanced controllers and compound actions are readily available for most collaborative robots and can be added to the knowledge base (see **Section 3.3.3** and **Table 2**). The integration of such new actions involves populating the ontology with instances and creating new relationships and conditions between them. Unfortunately, this is still a manual activity requiring core expertise on ontologies and their properties.

Recent and future developments in speech recognition might offer promising solutions to the mentioned problems in noisy environments. Neural networks and the utilization of other bio-signals (Schultz et al., 2017) are being developed with increasing recognition quality for individual words, as well as for natural language. This includes other sensor systems besides standard microphones, such as throat microphones, or neural devices.

Future work will combine computer vision and speech recognition for collaborative tasks. This allows for tasks that are more descriptive and can be better explained than pre-programmed. For example, a human operator could command the robot to hand over a tool with a red handle from a table with multiple colored tools. Such communication is well-suited to human cognitive skills, as it does not require much cognitive effort for object detection and requesting a command. For robots, on the other hand, such knowledge needs to be integrated beforehand by semantic annotation to the ontology and visual processing of camera images. In addition, feedback from speech recognition can make the system more explainable. Whenever the recognition of speech fails, incorrect words are used or an incorrect combination of words, a suitable command returned to the human would help in improving the collaboration. Similar troubleshooting procedures can be utilized for the reasoning over knowledge as well. Finally, future work will focus on user studies to analyse whether the collaboration is fluent and what concepts contribute to this, specifically targeting concepts such as efficiency, commitment and trust (Paliga and Pollak, 2021).

6 CONCLUSION

Coordination of shared tasks between a human and robot requires interaction modalities that are convenient, do not interfere with the task and can be adapted to new or changing situations. As including all possible scenarios and their outcomes into a robot action plan becomes easily intractable, this work enables a human to coordinate when and which robot actions are executed. Directing the robot is achieved by both a graphical user interface and a speech interface that takes known commands, in form of < `action,target` > -pairs, and transforms them into low-level actions. All information on actions, tasks and the world is stored in a knowledge base, which is utilized to track the actions and check whether selected actions are suitable or possible at the requested instance. The proposed system is evaluated by several industrial use cases, tested in a laboratory environment, where human-robot collaborative tasks require human coordination by command or speech. Results demonstrate that human coordination with simple commands is suitable to achieve and fulfill collaborative tasks in a fluent manner. Compared to the graphical interface, commanding by speech is preferred, as it does not require physical contact and attention stays with the shared task. On the other hand, noise and faulty speech recognition might prove to be problematic in real industrial environments. A thorough evaluation in real industrial environments, with tasks of similar complexity is, therefore, planned as future studies.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

AA, AS and RP designed the study. AA, AS and MN developed the methods and performed experiments. AA, AS, MN and RP analyzed the data and wrote the paper.

FUNDING

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 871449 (OpenDR).

REFERENCES

Angleraud, A., Houbre, Q., Kyrki, V., and Pieters, R. (2018). "Human-robot Interactive Learning Architecture Using Ontologies and Symbol Manipulation," in Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication, Nanjing, China, 27-31 Aug. 2018 (Nanjing, China: RO-MAN), 384–389. doi:10.1109/ROMAN.2018.8525580

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language Models Are Few-Shot Learners. arXiv preprint arXiv: 2005.14165

Chen, J. Y. C., and Barnes, M. J. (2014). Human-Agent Teaming for Multirobot Control: A Review of Human Factors Issues. *IEEE Trans. Human-mach. Syst.* 44, 13–29. doi:10.1109/thms.2013.2293535

Chen, Y., Xu, R., Lin, Y., and Vela, P. A. (2021). A Joint Network for Grasp Detection Conditioned on Natural Language Commands. arXiv preprint arXiv: 2104.00492

Chowdhury, A., Ahtinen, A., Pieters, R., and Vaananen, K. (2020). "User Experience Goals for Designing Industrial Human-Cobot Collaboration: A Case Study of Franka Panda Robot," in Proceedings of the 11th Nordic Conference on Human-Computer Interaction: Shaping Experiences (Tallinn, Estonia: Shaping Society), 1–13. doi:10.1145/3419249.3420161

Darvish, K., Simetti, E., Mastrogiovanni, F., and Casalino, G. (2021). A Hierarchical Architecture for Human-Robot Cooperation Processes. *IEEE Trans. Robot.* 37, 567–586. doi:10.1109/tro.2020.3033715

Dianatfar, M., Latokartano, J., and Lanz, M. (2021). Review on Existing VR/AR Solutions in Human-Robot Collaboration. *Proced. CIRP* 97, 407–411. doi:10.1016/j.procir.2020.05.259

Dukes, K. (2013). "Semantic Annotation of Robotic Spatial Commands," in Proceedings of the Language and Technology Conference (Dublin, Ireland: LTC). doi:10.3115/v1/S14-2006

Georgievski, I., and Aiello, M. (2015). HTN Planning: Overview, Comparison, and beyond. *Artif. Intelligence* 222, 124–156. doi:10.1016/j.artint.2015.02.002

Halme, R.-J., Lanz, M., Kämäräinen, J., Pieters, R., Latokartano, J., and Hietanen, A. (2018). Review of Vision-Based Safety Systems for Human-Robot Collaboration. *Proced. CIRP* 72, 111–116. doi:10.1016/j.procir.2018.03.043

High, R. (2012). The Era of Cognitive Systems: An inside Look at IBM Watson and How it Works. *IBM Corporation, Redbooks* 1, 16.

Hoffman, G. (2019). Evaluating Fluency in Human-Robot Collaboration. *IEEE Trans. Human-mach. Syst.* 49, 209–218. doi:10.1109/thms.2019.2904558

ISO-10218-1/2:2011 (2011). *Robots and Robotic Devices – Safety Requirements for Industrial Robots – Part 1: Robots/Part 2: Robot Systems and Integration. Standard*. Geneva, Switzerland: International Organization for Standardization.

ISO-15066:2016 (2016). *Robots and Robotic Devices — Collaborative Robots. Standard*. Geneva, Switzerland: International Organization for Standardization.

Johannsmeier, L., and Haddadin, S. (2017). A Hierarchical Human-Robot Interaction-Planning Framework for Task Allocation in Collaborative Industrial Assembly Processes. *IEEE Robot. Autom. Lett.* 2, 41–48. doi:10.1109/lra.2016.2535907

Kolbeinsson, A., Lagerstedt, E., and Lindblom, J. (2019). Foundation for a Classification of Collaboration Levels for Human-Robot Cooperation in Manufacturing. *Prod. Manufacturing Res.* 7, 448–471. doi:10.1080/21693277.2019.1645628

Kumar, S., Savur, C., and Sahin, F. (2021). Survey of Human-Robot Collaboration in Industrial Settings: Awareness, Intelligence, and Compliance. *IEEE Trans. Syst. Man. Cybern, Syst.* 51, 280–297. doi:10.1109/tsmc.2020.3041231

Liu, H., and Wang, L. (2018). Gesture Recognition for Human-Robot Collaboration: A Review. *Int. J. Ind. Ergon.* 68, 355–367. doi:10.1016/j.ergon.2017.02.004

Marin Vargas, A., Cominelli, L., Dell'Orletta, F., and Scilingo, E. P. (2021). Verbal Communication in Robotics: A Study on Salient Terms, Research fields and Trends in the Last Decades Based on a Computational Linguistic Analysis. *Front. Comput. Sci.* 2, 63. doi:10.3389/fcomp.2020.591164

Mavridis, N. (2015). A Review of Verbal and Non-verbal Human-Robot Interactive Communication. *Robotics Autonomous Syst.* 63, 22–35. doi:10.1016/j.robot.2014.09.031

Mi, J., Liang, H., Katsakis, N., Tang, S., Li, Q., Zhang, C., et al. (2020). Intention-related Natural Language Grounding via Object Affordance Detection and Intention Semantic Extraction. *Front. Neurorobot.* 14, 26. doi:10.3389/fnbot.2020.00026

Misra, D. K., Sung, J., Lee, K., and Saxena, A. (2016). Tell Me Dave: Context-Sensitive Grounding of Natural Language to Manipulation Instructions. *Int. J. Robotics Res.* 35, 281–300. doi:10.1177/0278364915602060

Müller-Abdelrazeq, S. L., Schönefeld, K., Haberstroh, M., and Hees, F. (2019). Interacting with Collaborative Robots-A Study on Attitudes and Acceptance in Industrial Contexts. *Social Robots: Technological, Societal and Ethical Aspects of Human-Robot Interaction*. Springer, 101–117. doi:10.1007/978-3-030-17107-0_6

Musen, M. A. (2015). The Protégé Project. *AI Matters* 1, 4–12. doi:10.1145/2757001.2757003

Ogenyi, U. E., Liu, J., Yang, C., Ju, Z., and Liu, H. (2021). Physical Human-Robot Collaboration: Robotic Systems, Learning Methods, Collaborative Strategies, Sensors, and Actuators. *IEEE Trans. Cybern.* 51, 1888–1901. doi:10.1109/tcyb.2019.2947532

Otter, D. W., Medina, J. R., and Kalita, J. K. (2022). A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Trans. Neural Networks Learn. Syst.* 32, 604–624. doi:10.1109/TNNLS.2020.2979670

Paliga, M., and Pollak, A. (2021). Development and Validation of the Fluency in Human-Robot Interaction Scale. A Two-Wave Study on Three Perspectives of Fluency. *Int. J. Human-Computer Stud.* 155, 102698. doi:10.1016/j.ijhcs.2021.102698

Ramirez-Amaro, K., Yang, Y., and Cheng, G. (2019). A Survey on Semantic-Based Methods for the Understanding of Human Movements. *Robotics Autonomous Syst.* 119, 31–50. doi:10.1016/j.robot.2019.05.013

Rocci, A., and Saussure, L. d. (2016). *Verbal Communication*. Berlin, Boston: De Gruyter.

Sadrfaridpour, B., and Wang, Y. (2017). Collaborative Assembly in Hybrid Manufacturing Cells: an Integrated Framework for Human–Robot Interaction. *IEEE Trans. Automation Sci. Eng.* 15, 1178–1192.

Savarimuthu, T. R., Buch, A. G., Schlette, C., Wantia, N., Roßmann, J., Martínez, D., et al. (2017). Teaching a Robot the Semantics of Assembly Tasks. *IEEE Trans. Syst. Man, Cybernetics: Syst.* 48, 670–692. doi:10.1109/TSMC.2016.2635479

Schultz, T., Wand, M., Hueber, T., Krusienski, D. J., Herff, C., and Brumberg, J. S. (2017). Biosignal-based Spoken Communication: A Survey. *Ieee/acm Trans. Audio Speech Lang. Process.* 25, 2257–2271. doi:10.1109/taslp.2017.2752365

Sharma, A. R., and Kaushik, P. (2017). "Literature Survey of Statistical, Deep and Reinforcement Learning in Natural Language Processing," in In Proceedings of the IEEE International Conference on Computing, Communication and Automation (Greater Noida, India: ICCCA), 350–354. doi:10.1109/ccaa.2017.8229841

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., et al. (2020). "Alfred: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 10740–10749. doi:10.1109/cvpr42600.2020.01075

Stenmark, M., and Malec, J. (2014). Describing Constraint-Based Assembly Tasks in Unstructured Natural Language. *IFAC Proc. Volumes* 47, 3056–3061. doi:10.3182/20140824-6-za-1003.02062

Sun, Z., Li, Z., Mu, Y., Song, S., Su, J., and Zhang, J. (2021). Intention Understanding in Human-Robot Interaction Based on Visual-NLP Semantics. *Front. Neurorobotics* 14, 121.

Taniguchi, T., Mochihashi, D., Nagai, T., Uchida, S., Inoue, N., Kobayashi, I., et al. (2019). Survey on Frontiers of Language and Robotics. *Adv. Robotics* 33, 700–730. doi:10.1080/01691864.2019.1632223

Thomason, J., Zhang, S., Mooney, R. J., and Stone, P. (2015). "Learning to Interpret Natural Language Commands through Human-Robot Dialog," in Proceedings of the International Joint Conference on Artificial Intelligence, 1923–1929.

Vanzo, A., Croce, D., Bastianelli, E., Basili, R., and Nardi, D. (2020). Grounded Language Interpretation of Robotic Commands through Structured Learning. *Artif. Intelligence* 278, 103181. doi:10.1016/j.artint.2019.103181

Villani, V., Pini, F., Leali, F., Secchi, C., and Fantuzzi, C. (2018b). Survey on Human-Robot Interaction for Robot Programming in Industrial Applications. *IFAC-PapersOnLine* 51, 66–71. doi:10.1016/j.ifacol.2018.08.236

Villani, V., Pini, F., Leali, F., and Secchi, C. (2018a). Survey on Human-Robot Collaboration in Industrial Settings: Safety, Intuitive Interfaces and Applications. *Mechatronics* 55, 248–266. doi:10.1016/j.mechatronics.2018.02.009

Williams, T., Briggs, G., Oosterveld, B., and Scheutz, M. (2015). "Going beyond Literal Command-Based Instructions: Extending Robotic Natural Language Interaction Capabilities," in Proceedings of the AAAI Conference on Artificial Intelligence, 1387–1393.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

F

SingleDemoGrasp: Grasping from a single image demonstration.

Amir Mehman Sefat[1,†], Alexandre Angleraud[1,†], Esa Rahtu[2] and Roel Pieters[1*]

December 15, 2021

Abstract

Learning-based grasping models typically require a vast amount of training data and training time to train an effective grasping model. Alternatively, small non-generic grasp models have been proposed that are tailored to specific objects by, for example, directly predicting the object's location in 2/3D space, and determining suitable grasp poses by post processing. In both cases, data generation is a bottleneck, as it has to be separately collected for each individual object. Moreover, most works consider objects in household scenarios. In this work, we tackle these issues and propose a light-weight grasping pipeline that is divided in four main steps: 1. single object demonstration, 2. object data augmentation, 3. grasp model training and 4. object grasping action. Four different vision-based methods are evaluated for deriving the relative rotation of the object with respect to the reference/target frame alongside an object detection module. Evaluation considers the grasping of different industrial and 3D printed objects with an industrial collaborative manipulator, and shows >90% success rate.

1 INTRODUCTION

Collaborative robots have gained popularity in industry as they are designed to be safe, particularly where human and robot share the workspace. Accompanied by intuitive programming interfaces, robot tasks can be programmed efficiently [1]. Despite the benefits, the application of cobots in industrial settings are mainly limited to offline tasks where the actions and targets are defined to the system beforehand [2, 3]. For example, in object pick and place tasks, the pose of the objects is fixed, and the robotic arm should go through a predefined trajectory to reach the exact grasping pose. Although there has been a great effort to mitigate these limitations by applying Artificial Intelligence (AI) and learning approaches [4], the accuracy of such methods is still not sufficient for industrial applications.

Grasping objects has been studied for a long time and with the advent of machine learning approaches and especially deep learning, researchers have been trying to develop stand alone models that are able to propose grasp poses for unseen objects. The most recent models are reported to have above 90% accuracy E.g. Dex-net 4.0 [5] is reported to achieve above 95% accuracy; However, most of the works has been done on generic household items where their complexity in terms of shape, texture and brightness and other parameters are totally different than the objects used in industrial scope which in many cases result in unsuccessful grasping attempts. Such learning-based models require vast amounts of training data and considerable training time on high-performance computing clusters. Consequently, grasping models are large in size and slow to execute.

In addition, most grasping models and datasets are limited to a specific object set or class type, such as household objects , industrial parts or warehousing items [6]. Extending datasets or retraining a grasping model is in most cases not an option, due to unavailable data or limitations in resources and

*[1]Unit of Automation Technology and Mechanical Engineering, [2]Unit of Computing Sciences, Tampere University, 33720, Tampere, Finland; †both authors contributed equally, `firstname.surname@tuni.fi`

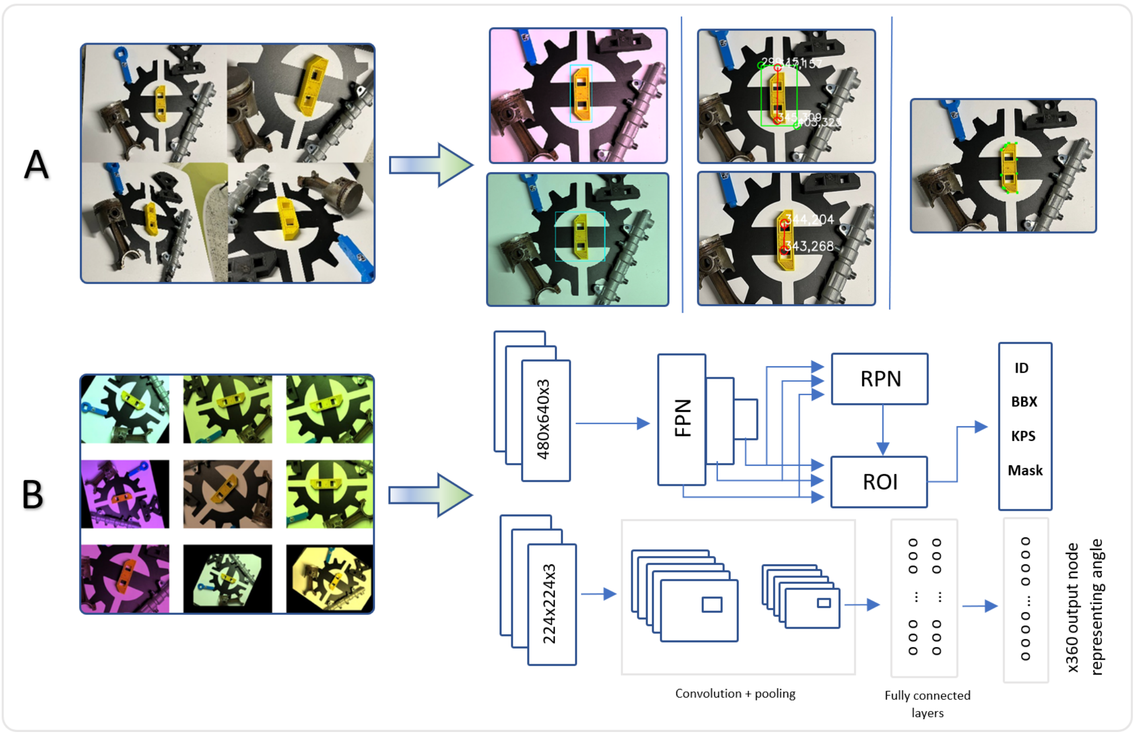


Figure 1: Should be replaced with actual figure

computation power. These problems exist in particular for small and medium sized (SME) companies, who do not have the knowledge and resources available for model training and fine-tuning.

In order to have an intelligent, meanwhile robust system to pick objects that are placed randomly in the workspace from a certain pose, De Coninck et al. [7] have suggested a hybrid model to combine neural network models that are capable of predicting location and orientation of an object followed by some pre-programming steps that results in a robust and successful grasp.

Inspired by [8], we propose an approach that utilizes state of the art deep learning approach "Faster-RCNN" [9] instead of CNN models to improve the efficiency of whole grasping pipeline. In this work, our grasping pipeline generates an augmented training dataset from a single user demonstration which consist of a few images of the object from different camera views. These images then go through data augmentation process to generate training data which are then utilized to train a grasping model.

Our main observation to motivate this work is that collecting or generating training data for industrial parts is a tedious, time-consuming and costly task, which is often out of reach for industrial stakeholders. Even though plenty datasets can be found, each are limited (to some extend) to the objects they contain. Industrial SMEs require the handling of objects that, in most cases, do not resemble objects in these datasets, or the objects themselves can change depending on a customer's requirement. In addition, SMEs often do not have the knowledge and computation infrastructure to train novel grasp detectors, even if object models are available.

In this work, different variats of faster-rcnn architecture using Detectron2 [10] models are evaluated to identify the best representation for specific object properties, i.e., shape complexity, texture and for different environmental conditions. Results are demonstrated in simulation (Gazebo and Webots) and real experiments (Franka Panda with standard gripper) with an industrial object set are performed to study the performance of our proposed pipelines.

The main contribution of our work are as follows:

- 3D planar grasp model for industrial parts
- data generation by augmenting a single object demonstration
- training applicable networks
- evaluation of the approach in simulation and with real experiments

The paper is organized as follows: Section 2 reviews related works and state of the art in computer vision and grasping methods with respect to neural networks. Section 3 and Section 4 define the considered problem statement and research methodology, respectively. Section 5 describes the implementation details of the proposed grasping pipeline, and Section 6 reports the results and provides a comparison to the state of the art. Finally, Section 7 concludes the work.

2 Related Work

In the context of grasping, Object detection, pose estimation and grasp detection are closely related, as grasp poses or grasp actions can be directly generated from an object pose. This section presents a brief overview of related approaches.

2.1 Object detection and pose estimation

Traditionally, object detection and pose estimation algorithms have utilized classical 2D features that exploit local salient details, such as corners, edges and ridges. Well-known detectors like SIFT [11], SURF [12] and ORB [13] can extract robust keypoints from a scene by relying on texture on objects or of the scene itself. Texture-less keypoint detection, on the other hand, utilizes geometrical primitives as features in methods such as BOLD [14], BORDER [15] and BIND [16]. In addition, alternatives to traditional keypoints are template matching, where a image patch provides the template to localize within an image, or deep features that extract keypoints based on high-level cues captured by convolutional neural networks. The latter is a recent development that has gained popularity due to their data-driven property and promising performance [17], as compared to hand-crafted features. Analogous to 2D keypoints for RGB images, 3D keypoints can be extracted from 3D data representations, such as pointclouds or volumetric images [18]. Following the detection of keypoints from a raw image, follow-up steps include the description of the keypoint and the matching of them over two or multiple images.

In a similar manner, Convolutional Neural Network (CNN) based detectors, such as YOLOV3 [19] or Faster R-CNN [9], could be utilized to detect objects, after which their orientation should be computed to obtain a planar grasp candidate.

Object pose estimation on the other hand can be done using three different approaches [20].

- correspondence-based methods such as LCD [21] and 3DMatch [22].
- Template-based 6D object pose estimation methods such as PoseCNN [23].
- Voting based methods such as PVN3D [24] and DenseFusion [25].

2.2 Grasp detection

Object grasp detection is a popular topic in robotics and can be divided in several categories to differentiate between approaches and their assumptions. For example, the representation of a grasp is an important consideration and determines the complexity of the problem and its application. When considering only a planar grasp pose representation, grasp detection is simplified to finding the object

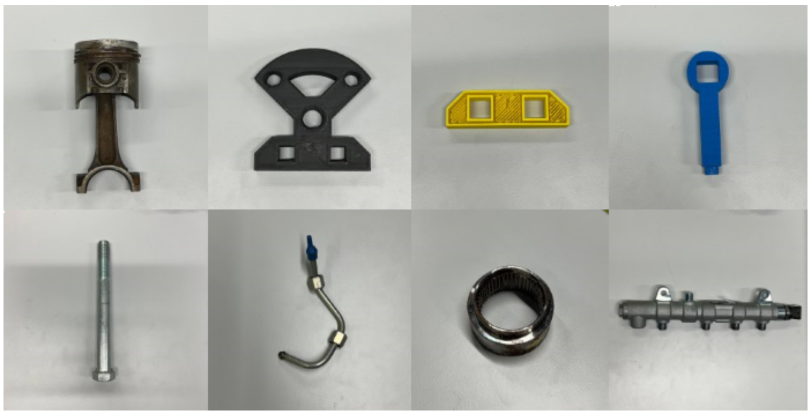


Figure 2: objects used for evaluation of grasping pipelines

and its orientation on a planar surface, typically represented as an (oriented) bounding box. On the other hand, in case a complete 3D pose of the object is required for grasping, detection should return the full 3D position and 3D orientation of the object. In context of learning-based grasp detection, typical data-driven approaches differentiate between the utilization of RGB, depth (in form of pointclouds) or a combination of both (RGB-D). In addition, objects to be grasped can be known, similar (i.e., different instance of a known category) or novel, which should be considered when deciding (or developing) on the data representation, collection and training approach. A comprehensive survey of approaches is presented in [20].

2.3 Datasets

Existing datasets for 2D object detection, such as Pascal VOC [26], COCO [27] and, more recently, Objectron [28] for 3D objects, are widely available, including common objects that are present in everyday scenes. There are also datasets designed more specifically such as EGAD! by creating 3D meshes with diverse properties [29] to cover variations in object properties and also ACRONYM [30] where the collected data are from simulation physics.

These publicly available datasets are although makes it possible to have a generic dataset and include different categories of object and makes it possible to have a reasonable comparison and evaluation of the performance, they are not suitable for applications where the target application is specific to a particular group of objects. On the other hand, they are not easily extendable as this is an expensive process in terms of time and resources.

2.4 Robot Grasping Solutions

The methods explained in the previous sections generates grasp pose outputs that have a specific format that are defined in 2D or 3D and there are robustness factors defined to measure the accuracy of such models [3]. Such methods require motion planning to send the end effector to the desired grasp pose from the correct trajectory to achieve a successful grasp. The motion planning approaches for this purpose can be listed in a very general manner as DMP-based methods, Imitation learning and reinforcement learning methods [20]. On the other hand, There are approaches to develop a visuomotor controller to directly generate robot motion from inferred input [31] for example using reinforcement learning [20].

3 Problem Statement

3.1 Object Grasping Scenario

The robot object grasping scenario considers a robot manipulator with standard gripper and objects that are located on a planar table in front of it. Objects originate partly from a Diesel engine assembly use case and have both simple and complex geometry (see Fig. 3). All objects should allow for a stable grasp, without alteration to the gripper or object pose and be light enough to be lifted (< 1 kg). As general rule, we denote that each object can be represented by a 3D planar position and orientation $\{x, y, \theta_o\}$, from which a grasp pose is extracted. The grasping problem can then be stated as follows: from a single object image demonstration, generate suitable training data to train a grasp detection model that can run in real-time to successfully grasp new and unknown objects.

In this scenario, eight different objects (see fig. 2) are selected from a real diesel engine and also four 3D-printed objects that are different in terms of mass distribution, texture, symmetry and scale that describes the general properties of objects in industrial settings. For example, fuel line from the engine block has a small width and a non symmetric shape with even mass distribution while a piston has a symmetric shape, low aspect ratio meanwhile an uneven mass distribution. However, to simplify the problem, the objects are selected carefully to be distinguishable from different views and the grasp locations are reachable easily by the end-effector.

3.2 Planar and 3D grasp representation

The grasping model should produce an grasp pose in the form of a grasp location with a relative orientation represented as $\hat{b} = \{x, y, \theta_o\} \in \mathbb{R}^3$ that defines the grasp pose in planar space. In 3D world space, A rigid pose of the gripper, connected to the end-effector of the robot, is defined as $\mathbf{T}_g = (\mathbf{R}_g, \mathbf{t}_g) \in SE(3)$.

In addition to the grasp pose, two intermediate poses are defined to assist in the grasping sequence: a hover pose $\mathbf{T}_h = (\mathbf{R}_h, \mathbf{t}_h) \in SE(3)$ and a pre-grasp pose $\mathbf{T}_p = (\mathbf{R}_p, \mathbf{t}_p) \in SE(3)$. Hover pose in fact represent an imaginary pre-grasp pose after which the actual grasp is possible by going through a pre-defined trajectory. On the other hand, the pre-grasp pose is defined as an offset with respect to grasp pose in z-direction to avoid collision with the object.

The translation from the planar output of the detection module to world frame is done using the mathematical equations 1 and 2 that holds in pin-hole camera model after camera calilbration.

$$X = x\frac{Z}{f}, \tag{1}$$

$$Y = y\frac{Z}{f}, \tag{2}$$

where X, Y, Z correspond to world coordinates and x, y denote pixel coordinate and f is the focal length of the camera.

4 Methodology

The proposed method is separated into a baseline grasping approach, complimented by four different perception models, to compute a planar grasp pose. The grasping approach (Fig. 1) consists of the following four distinct steps:

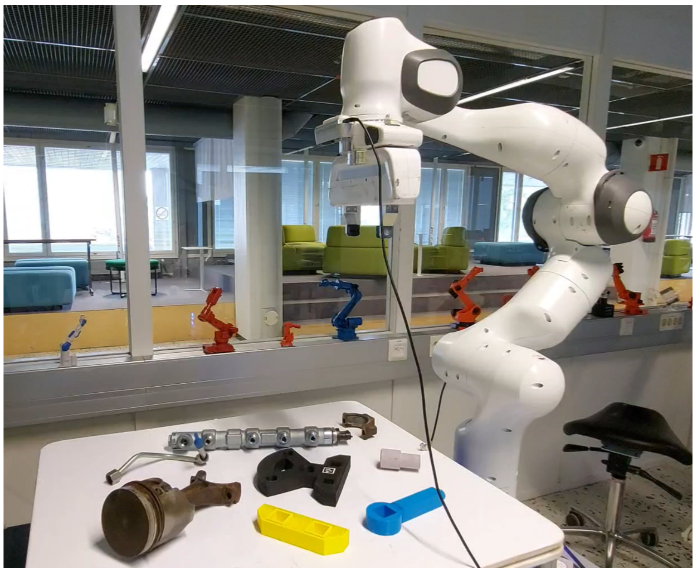


Figure 3: Object grasping setup.

Table 1: Object grasp detection model. Abbreviations: BB - bounding box, KP - keypoints, CL - class label.

	Object position estimation		**Object orientation estimation**			
Grasping model	**Pre-trained model**	**Output format**	**Pre-trained model**	**Method**	**Human input**	**Training data**
A	Faster R-CNN	BB	CNN	Classification	Object BB	BB, cropped box
B	Keypoint R-CNN	BB and KP	Keypoint R-CNN	KP	Object BB, mask and 2 KP	BB, KP, CL
C	Keypoint R-CNN	KP	Keypoint R-CNN	KP		KP, CL
D	Mask R-CNN	BB and object mask	Mask R-CNN	mask + SIFT		BB, Mask, CL

1. **Human input** - captures and annotates the object in the field of view of the camera.
2. **Training data** - is generated automatically by applying data augmentation techniques.
3. **Object grasp pose** - is estimated based on different state of the art neural networks.
4. **Grasping action** - is done after converting planar grasp to 6D pose.

Following, we describe the four different combinations of perception modules.

4.1 Object detection alongside a CNN for predicting orientation

The object grasp location and orientation are estimated separately by training a Faster R-CNN and a basic CNN, respectively.

The Faster R-CNN network takes pairs of images and their corresponding annotations (bounding boxes) as input for training, and generates a bounding box around the objects if they are present in the image scene. The input images are simply annotated by defining a bounding box around the object followed by data augmentation. The architecture of the Faster-RCNN model and the input format are illustrated in Fig. 1.

In order to predict the relative orientation, a simple CNN network is implemented where the final layer consist of 360 output node to represent the class of the object's orientation that limits the precision to one degree which is enough in this case for our grasping pipeline.

Table 2: Predictors' used hyper-paramethers. Abbreviations: lr: learning rate, iters: iterations, ipb: image per batch, num_cls: number of classes

	A	B	C	D
Object Detector	lr: 0.005 iters: 500 ipb: 8 num_cls: 1	lr: 0.0008 iters: 1000 ipb: 2 num_cls: 1	lr: 0.0008 iters: 1000 ipb: 2 num_cls: 1	lr: 0.008 iters: 1000 ipb: 2 num_cls: 1
Orientation Predictor	loss: categorical_crossentropy epochs: 15 batch_size: 32 optimizer: adam num_cls: 360	Not applicable		

The first layer of this CNN network accepts image arrays with a shape of $(224 \times 224 \times 3)$ to extract features and classifies the object based on the highest score to predict the corresponding orientation. To generate training data, the same procedure is followed as explained for Faster-RCNN method, with the difference that at each step of augmentation, the bounding boxes are used to crop and resize the region of interest that are labeled with the corresponding orientation.

In order to keep track of the network's performance, the following angle error metric is used:

$$\theta_e = \||\|\theta_{true} - \theta_p\| - 180\|, \tag{3}$$

where θ_{true} is the true angle and θ_p is the predicted angle.

4.2 grasp pose extraction using Keypoint detection

In contrast with the previous pipeline, here a single network (Keypoint- RCNN) which is a variant of Faster-RCNN network is utilized to detect both object and keypoints. During input annotation in this pipeline, 12 keypoint on the object is defined carefully on a straight line that represent the reference orientation.

Finally, The bounding box center is used as the planar hover position and the relative orientation is extracted from the keypoints to form the planar grasp pose as follows: $\mathbf{t}_h = (\mathbf{x}, \mathbf{y}, \theta)$ which is then converfted to 6D pose for the final grasping action.

As a result, the proposed pipeline now only comprises of one stage of augmentation and one stage of prediction.

4.3 Keypoint detection for proposing direct grasp poses

While the keypoint detector module demonstrated a greater stability and robustness in terms of successful grasps, one could improve the pipeline by directly assuming the center of keypoints to be the center of grasp pose.

For this purpose, the same network mentioned in B, is used where in this case the pipeline only utilizes the keypoints themselves to localize the "grasp location" instead of bounding boxes' center and the relative orientation is calculated similar to B.

Having the information about location and orientation of the grasp pose, the system could directly transfer the 2D coordinates to 3D from camera frame to the world frame in order to directly pick the object and there is no more need to reach the hover pose as the end-effector can be directly sent to the grasp pose which simplifies the baseline model to a great extend and also shows more stable results as illustrated in table 3.

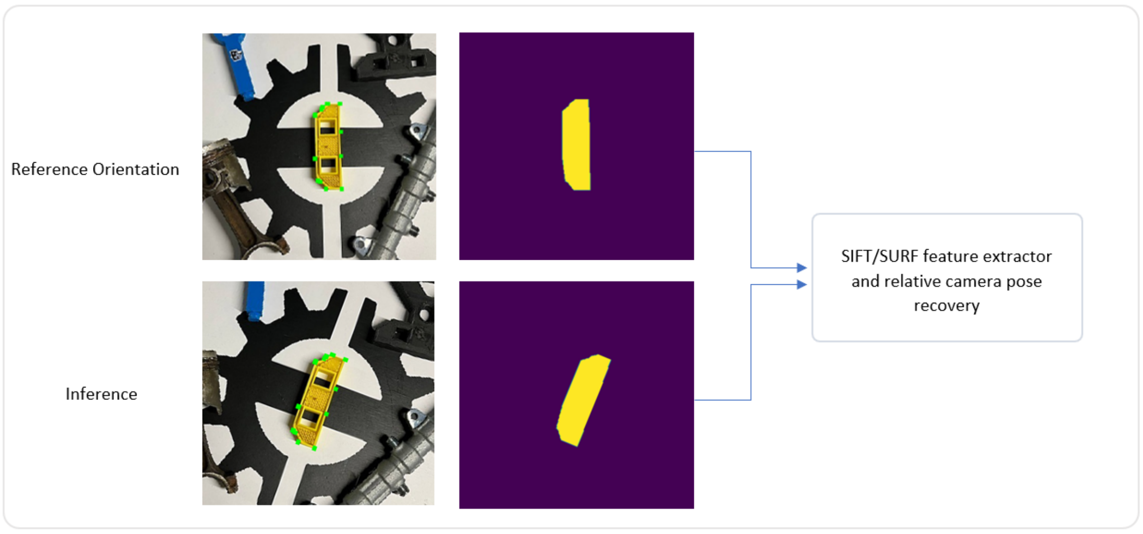

Reference Orientation
Inference
SIFT/SURF feature extractor and relative camera pose recovery

Figure 4: recovering camera pose using predicted mask and local feature extractors

4.4 grasp pose extraction using mask prediction

Object mask estimation can return a planar object position and orientation based on predicted mask using Mask R-CNN architecture. This is possible, as the grasping model only requires 2D object information, based on the object mask that separates the object from the background.

Mask-RCNN again is a variant of Faster-RCNN which is designed to detect object on the image and predict a mask (polygon) over the object. To train such network, it is required to prepare the training data such that it include both bounding box around the object and also a list of keypoints that construct the mask around the object as shown in Fig. 1

One additional step is necessary to determine the grasp orientation, which in case of this study is done by converting the mask over the object to a binary image followed by local feature extraction methods to estimate the relative orientation as shown in Fig. 4.

For this purpose, the detection part consist of the following steps:

1. mask detection network generate mask over objects.
2. the generated mask is filtered to extract geometrical features of the object.
3. the extracted binary mask is now being used to find local feature based and consequently relative rotation between the reference binary mask and extracted binary mask from the prediction.

4.5 Augmented Dataset Generation

Dataset generation utilizes a single input image (RGB, $480 \times 640 \times 3$) taken above the workspace in which the target object is visible. These images are then annotated as illustrated in Fig. 1 with respect to their corresponding pipeline input format.

Augmentation of the reference image is done as follows: First, the image is annotated and labeled by defining the required bounding boxes or keypoints over the object and a sequence of common augmentation techniques are performed, i.e., cropping, zooming, rotation, translation, etc. Finally, the data-augmentor automatically translates all the annotations from the input image to the augmented images accordingly.

During each data augmentation process, 1500 training samples and 200 validation samples are generated, for position and orientation data, respectively. For the CNN network of first pipeline, the

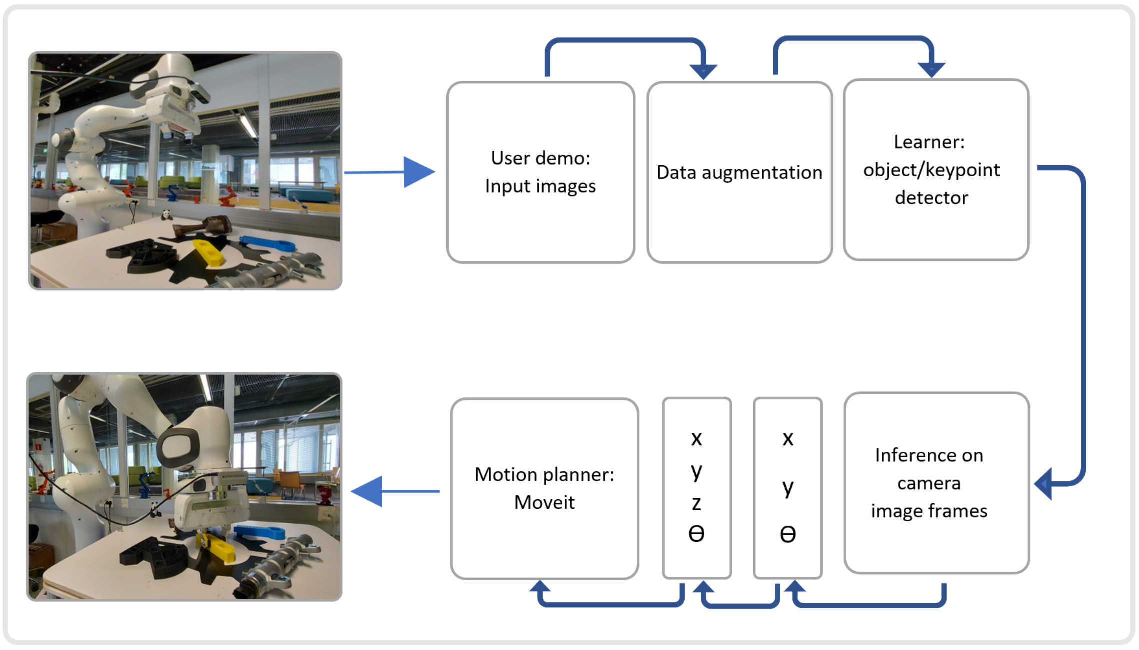


Figure 5: schematic representation of the grasping architecture.

number of generated samples for training angle predictor is 5000. More details about each pipeline can be found in table 3.

5 Implementation

Details concerning the implementation of the proposed grasping models are described in the following section and includes the perception module, robot motion generation as well as simulation.

5.1 Architecture

The grasping pipeline is divided into two major computation nodes, which are explained as follows.

Perception - feeds the neural network models with the input images, runs inference and generates an output. The raw output of the models as explained in section 4 are then used to calculate the planar grasp pose of the object with respect to the image plane. The 2D information is then translated into 3D coordinates in the world frame and sent to the motion controller node. The translation from 2D to 3D is done using equations 1 and 2 that holds in pin-hole camera models when all the intrinsic and extrinsic parameters of the camera are available. The structure of the perception module is illustrated in Fig. 1.

Motion control - generates the actions and motions of the robot manipulator and gripper in order to execute a grasp. It receives input from the perception node, and directly commands a grasping action. Motion generation is done using ROS MoveIt[1], with a Cartesian position controller running on the robot at 1000 Hz.

The overall structure of the single demonstration grasping is illustrated in figure 5.

[1]`https://moveit.ros.org`

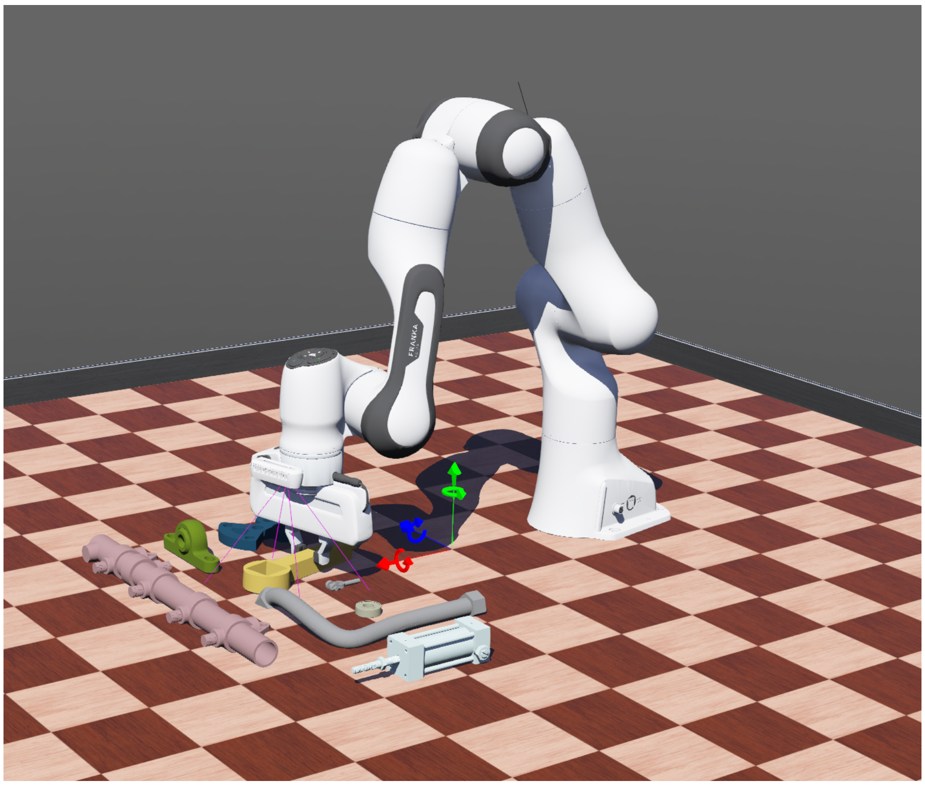


Figure 6: Implementation of grasping pipeline in Webots simulator

5.2 Grasp Detection Network

All the pipelines utilize Detectron2 [10] as their perception module. The generated training data and their corresponding labels are fed to the learner, according to the corresponding pipeline input format. Table 1 and 2 summarizes the information regarding the implementation and specification for perception module of each individual pipeline.

The detection networks as mentioned previousely, utilize Faster-RCNN or one of its variants depending on the application. Similar to generic use cases where networks are designed to be trained for multiple object classes, one can take benefit of that by developing such generic detector, however it would require larger datasets and higher training times. In this work, the pipeline is developed such that each demonstration result in a completely new and unique detector for each individual object. For this purpose, only 1500 training samples is sufficient accompanied by 200 validation samples.

All the models as explained in 4 have as output a bounding box, where the bounding box center represents the hover position, except for 4.3 where the hover pose is not utilized and the center of the keypoints is taken as the grasp position.

Finally, the pipelines learn how to find grasp locations and orientations by post processing the outputs of trained networks, followed by corresponding robot action as explained in 3.2.

5.3 Simulation environment

For quick evaluation of the developed grasp models, two simulation environment containing all relevant steps are implemented. This enables grasping models to be assessed without costly robotic hardware, speeding up developments considerably. For this purpose, 3D models of all objects are obtained to have a better idea on performance and limitations when doing real experiments. Both Gazebo and

Webots (see figure 6) are utilized to demonstrate the functionalities, and are provided freely available to the research community.

6 Results and Comparison

Experimental results, their analysis and a comparison to other related work follows in this section.

To evaluate the performance of studied pipelines, the grasping scenario is performed in simulation environment and extensive grasp experiments, executed with a collaborative robot (Franka Emika Panda), RGB-D camera (Intel Realsense D-435) and standard gripper (see Fig. 3). All objects are placed at random configurations on the table in front of the robot and 10 robot grasp attempts are executed for each object from different robot starting configurations.

Table 3 illustrates the performance of each developed pipelines on average, on all the objects. The success rate in this study translates to the percentage of successful grasping attempts with respect to total attempts.

During the experiments, random objects are also placed in the view of the camera to see the affect of unseen and similar objects where the detector performed perfectly as long as the optimal score threshold and hyper parameters are being used. However, illumination and brightness of the environment still is a major affecting factor in this work and generally in vision applications.

6.1 Grasp Detection Results

The first pipeline turns the learning process into a two-stage detector where two separate training is needed and during inference, two consecutive predictions are needed to get location and orientation of the object, which makes the pipeline computationally inefficient. On the other hand, another dataset must be generated to train the convolutional neural network. Although this model showed above 90% success rate, having to train two different network bring extra complexity to the system and the CNN angle predictor shown higher error comparing to keypoint detection module. Moreover, the high success rate reported during the experiments is due to the fact that faster-rcnn showed a more robust bounding box detection performance than the keypoint detectors in terms of finding Minimum Area Rectangle (MAR).

The second pipeline utilizes keypoint-rcnn architecture and the outputs are directly used to derive planar grasping pose. This pipeline simplifies the pipeline as there is less effort needed by the user for input demonstration. However, still user is required to demonstrate a final grasping action to the system from hover pose to pre-grasp pose. On the other hand, although the detection accuracy of keypoint-rcnn shows similar performance to faster-rcnn, minimal errors observed in the MAR prediction were the main reason of lower success rate with respect to the first pipeline.

Pipeline C showed the highest success rate among all the pipelines. Although the same network from pipeline B is used for training, however the minimal change in the input to the network mitigates the accuracy issues of the second pipeline by directly using keypoints center instead of bounding boxes' center as the grasping planar position. Not only this makes the prediction more accurate, it also simplifies the grasping system as the final grasp pose can directly be reached by translating the 2D coordinate of grasp location to 3D as explained in 3.2.

Implementing pipeline D was with the objective of directly retrieving the 3D camera relative pose with respect to the input demonstration. However, this requires a huge number of features on the scene which makes the problem complicated when the shapes share similarities with each other. The annotation of input image is much complicated and time consuming as the user have to draw a mask/polygon over the object which is not consistent with the final objective of this work to develop a user friendly grasping pipeline.

Inference performance of the pipeline's detection modules are measured on a desktop and laptop GPU (Nvidia GTX 1080 Ti and GeForce 940mx). Then computational performance of the pipelines in terms of training data, data size, training time, inference time etc. are illustrated in 3.

Table 3: grasping results in terms of success rate, training performance and inference speed

Grasping success rate		Inference speed (FPS)	
Grasping pipeline	Average success rate, 80 attempts	GTX 1080 ti	Geforce 940mx
A	0.9125	20	2.5
B	0.825		
C (simulation)	0.935		
C (real experiment)	0.8875		
D	0.775		
Training			
Dataset size	training time (average)	model size	
1500 (faster rcnn), 5000 (CNN)	00:14:00 00:02:00	faster-rcnn: 300 mb CNN: 8 mb	
1500	00:07:30	450 mb	
1500	00:07:00	450 mb	
2000	00:08:00	450 mb	
1500	00:06:30	330 mb	

Other than the characteristics of each pipeline, there are also other factors that highly affect the performance of pipelines. For example, shape complexity, texture, color and brightness of the environment still have effect on the performance as other 2D RGB vision applications. Due to the same reason, as depth information is not available, grasping height is also must be known prior to grasping which is another limitation of developed grasping modules.

It is worth mentioning that in an experiment with pipeline C, 7 different views of the object was demonstrated to the system that dramatically increased the robustness of detection in different environmental conditions.

7 CONCLUSIONS

Based on the results from the previous section, pipeline C has shown the most robust grasping method where both in simulation and real experiments, demonstrate a high success rate and reliability compared to the other tested methods. On the other hand, not only it requires less computational resources, but also eliminates the need for the user to demonstrate the final grasping movement which makes it more intuitive, simple, and faster. Although the proposed model still is limited to known objects and planar movements, still it can extend the freedom of objects by mitigating the need for fixed object locations which can be highly effective in human-robot-collaborative tasks.

Acknowledgements

Project funding was received European Union's Horizon 2020 research and innovation programme, grant agreement no. 871449 (OpenDR).

References

[1] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, pp. 248–266, 2018.

[2] S. El Zaatari, M. Marei, W. Li, and Z. Usman, "Cobot programming for collaborative industrial tasks: An overview," *Robotics and Autonomous Systems*, vol. 116, pp. 162–180, 2019.

[3] E. Matheson, R. Minto, E. G. Zampieri, M. Faccio, and G. Rosati, “Human–robot collaboration in manufacturing applications: A review,” *Robotics*, vol. 8, no. 4, p. 100, 2019.

[4] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, “A survey on learning-based robotic grasping,” *Current Robotics Reports*, pp. 1–11, 2020.

[5] J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, and K. Goldberg, “Learning ambidextrous robot grasping policies,” *Science Robotics*, vol. 4, no. 26, 2019.

[6] J. Mahler, M. Matl, X. Liu, A. Li, D. Gealy, and K. Goldberg, “Dex-net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1–8.

[7] E. De Coninck, T. Verbelen, P. Van Molle, P. Simoens, and B. Dhoedt, “Learning to grasp arbitrary household objects from a single demonstration,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2372–2377.

[8] ——, “Learning robots to grasp by demonstration,” *Robotics and Autonomous Systems*, vol. 127, p. 103474, 2020.

[9] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *arXiv preprint arXiv:1506.01497*, 2015.

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

[11] D. G. Lowe, “Object recognition from local scale-invariant features,” in *IEEE international conference on computer vision (ICCV)*, vol. 2, 1999, pp. 1150–1157.

[12] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded up robust features,” in *IEEE European Conference on Computer Vision (ECCV)*, 2006, pp. 404–417.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” in *International conference on computer vision (ICCV)*, 2011, pp. 2564–2571.

[14] F. Tombari, A. Franchi, and L. Di Stefano, “BOLD features to detect texture-less objects,” in *IEEE International Conference on Computer Vision (ICCV)*, 2013, pp. 1265–1272.

[15] J. Chan, J. Addison Lee, and Q. Kemao, “Border: An oriented rectangles approach to texture-less object recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2855–2863.

[16] ——, “Bind: Binary integrated net descriptors for texture-less object recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2068–2076.

[17] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, “Image matching from handcrafted to deep features: A survey,” *International Journal of Computer Vision*, vol. 129, no. 1, pp. 23–79, 2021.

[18] F. Tombari, S. Salti, and L. Di Stefano, “Performance evaluation of 3d keypoint detectors,” *International Journal of Computer Vision*, vol. 102, no. 1-3, pp. 198–220, 2013.

[19] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

[20] G. Du, K. Wang, S. Lian, and K. Zhao, “Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review,” *Artificial Intelligence Review*, pp. 1–58, 2020.

[21] Q.-H. Pham, M. A. Uy, B.-S. Hua, D. T. Nguyen, G. Roig, and S.-K. Yeung, "LCD: Learned cross-domain descriptors for 2D-3D matching." in *AAAI*, 2020, pp. 11 856–11 864.

[22] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from RGB-D reconstructions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1802–1811.

[23] C. Capellen, M. Schwarz, and S. Behnke, "Convposecnn: Dense convolutional 6D object pose estimation," *arXiv preprint arXiv:1912.07333*, 2019.

[24] Y. He, W. Sun, H. Huang, J. Liu, H. Fan, and J. Sun, "PVN3D: A deep point-wise 3D keypoints voting network for 6DOF pose estimation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 632–11 641.

[25] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6D object pose estimation by iterative dense fusion," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3343–3352.

[26] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *IEEE European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 740–755.

[28] A. Ahmadyan, L. Zhang, J. Wei, A. Ablavatski, and M. Grundmann, "Objectron: A large scale dataset of object-centric videos in the wild with pose annotations," *arXiv preprint arXiv:2012.09988*, 2020.

[29] D. Morrison, P. Corke, and J. Leitner, "EGAD! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation," *IEEE Robotics and Automation Letters*, 2020.

[30] C. Eppner, A. Mousavian, and D. Fox, "ACRONYM: A large-scale grasp dataset based on simulation," *arXiv preprint arXiv:2011.09584*, 2020.

[31] S. Caldera, A. Rassau, and D. Chai, "Review of deep learning methods in robotic grasp detection," *Multimodal Technologies and Interaction*, vol. 2, no. 3, p. 57, 2018.