Interner Bericht

DLR-IB-RM-OP-2023-157

Integration of Continual Learning and Semantic Segmentation in a vision system for mobile robots

Masterarbeit

C. David Echeverry Valencia

Deutsches Zentrum für Luft- und Raumfahrt
Institut für Robotik und Mechatronik
Oberpfaffenhofen

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Aalto University
School of Electrical
Engineering

Master's programme in Space Science and Technology

Integration of Continual Learning and Semantic Segmentation in a vision system for mobile robotics

C. David Echeverry Valencia

Master's Thesis
2024

Aalto University
School of Electrical
Engineering

Author C. David Echeverry Valencia

Title Integration of Continual Learning and Semantic Segmentation in a vision system for mobile robotics

Degree programme Space Science and Technology

Major Space Robotics and Automation

Supervisor Prof. Pekka Marttinen

Advisor Mr Jongseok Lee

Collaborative partner Deutsches Zentrum für Luft- und Raumfahrt

Date January 22, 2024 **Number of pages** 87+8 **Language** English

Abstract

Over the last decade, the integration of robots into various applications has seen significant advancements fueled by Machine Learning (ML) algorithms, particularly in autonomous and independent operations. While robots have become increasingly proficient in various tasks, object instance recognition, a fundamental component of real-world robotic interactions, has witnessed remarkable improvements in accuracy and robustness. Nevertheless, most existing approaches heavily rely on prior information, limiting their adaptability in unfamiliar environments.

To address this constraint, this thesis introduces the Segment and Learn Semantics (SaLS) framework, which combines video object segmentation with Continual Learning (CL) methods to enable semantic understanding in robotic applications. The research focuses on the potential application of SaLS in mobile robotics, with specific emphasis on the TORO robot developed at the Deutsches Zentrum für Luft- und Raumfahrt (DLR). Evaluation of the proposed method is conducted using a diverse dataset comprising various terrains and objects encountered by the TORO robot during its walking sessions.

The results demonstrate the effectiveness of SaLS in classifying both known and previously unseen objects, achieving an average accuracy of 78.86% and 70.78% in the CL experiments. When running the whole method in the image sequences collected with TORO, the accuracy scores were of 75.54% and 84.75%, for known and unknown objects respectively. Notably, SaLS exhibited resilience against catastrophic forgetting, with only minor accuracy decreases observed in specific cases. Computational resource usage was also explored, indicating that the method is feasible for practical mobile robotic systems, with GPU memory usage being a potential limiting factor.

In conclusion, the SaLS framework represents a significant step forward in enabling robots to autonomously understand and interact with their surroundings. This research contributes to the ongoing development of robotic systems that can operate effectively in unstructured environments, paving the way for more versatile and capable autonomous robots.

Keywords Continual Learning, Progressive Neural Networks, mobile robotics, Computer Vision, Machine Learning, Semantic Segmentation

Preface

I would like to thank my advisor, Jongseok Lee, for his kind and constant guidance through all the thesis, and my supervisor, Pekka Marttinen, for his attention during these months. I would also like to acknowledge the TORO team: Jinoh Lee, George Mesesan and Robert Schuller, for finding the time to help us collect data from the robot.

También me gustaría expresar profunda gratitud a mis padres, María y Alberto, por haberme brindado su constante apoyo a través de mis años de estudio y del proceso de esta tesis.

Finalmente, agredecer a mi novia, Nada, por su cariño y motivación constante. Gracias por todo.

C. David Echeverry Valencia

Contents

Abbreviations

AOT	Associating Objects with Transformers
AI	Artificial Intelligence
CL	Continual Learning
CoDEPS	Continual Learning for Depth Estimation and Panoptic Segmentation
COMFORMER	Continual MaskFormer
CV	Computer Vision
DeAOT	Decoupling features in Hierarchical Propagation
DL	Deep Learning
DLR	Deutsches Zentrum für Luft- und Raumfahrt
GPU	Graphics Processing Unit
INSTR	Instance Stereo Transformer
IROS	International Conference on Intelligent Robots and Systems
LwL	Learning What to Learn
MSP	Maximum Softmax Probability
ML	Machine Learning
NN	Neural Network
OOD	Out-of-Distribution
PLOP	Pseudo-label and Local Pooled Outputs Distillation
PNN	Progressive Neural Networks
RECALL	Replay in Continual Learning
SAM	Segment Anything Model
SAM-Track	Segment and Track Anything
SaLS	Segment and Learn Semantics
VOS	Video Object Segmentation

1 Introduction

The last decade has increasingly witnessed the use of robots in many applications, specifically those in which machines are able to operate in an independent and automatic manner. Their performance has increased exponentially by combining new Artificial Intelligence (AI) methods and Machine Learning (ML) algorithms, leading to new tools even more precise than humans.

For instance, robots engaging with humans in real-world settings typically encounter a wide range of object instances. The field of object instance recognition, which has made significant advancements in terms of accuracy and robustness, partially addresses the issue of acquiring the information required for successful interaction with these objects. However, most current approaches demand prior information in the form of 3D models or annotated data for each object class or instance under consideration.

To address this limitation, a method has been developed for avoiding the need for prior information about the environment: Segment Anything Model (SAM) [31]. This new tool has been introduced by Meta AI as a model for image segmentation, which has been trained with more than one billion masks to enable Computer Vision (CV) applications. Such algorithm opens the field to a more independent automatic approach, which has been recently explored in several robotic applications, such as those related to planetary exploration. One of its main limitations is that it cannot segment objects in videos, but it is solved with Segment and Track Anything (SAM-Track) [11], which joins a powerful tracking method with SAM in order to obtain a real-time instance segmentation.

However, little research has been devoted to learning the semantics occurring after segmentation of unknown objects using an algorithm for detecting and then classifying object instances into known categories while learning new unknown classes. Although SAM can currently segment objects of unknown classes, they lack the semantics often needed for high-level robot decision making and autonomy in the context of planetary exploration. For example, if a rover equipped with a robotic arm were assigned a mission to pick up a specific type of rock 'Granite', executing this task would require that the robot understand the semantics of rocks in its environment.

A potential solution for overcoming this issue would be to integrate current object segmentation methods with Continual Learning (CL) methods in order to infer semantic information based on an algorithm that is able to detect object instances and then classify them into known categories while learning new unknown classes. These methods are based on Neural Networks (NN) that hold and accumulate knowledge over different tasks, solving what is called "catastrophic forgetting", the inability of a NN to remember previously learned information [13, 36]. One of the multiple approaches that have been presented for ML is the Progressive Neural Networks (PNN), a model architecture that retains trained networks for different tasks and allows a learning transfer between them [47].

Some works have attempted to to integrate both segmentation and learning methods for detecting and classifying unknown objects. Most of them have been released in the last three years, so it is remarkable that this field of Continual Semantic Segmentation

is still being developed. For instance, Continual MaskFormer (CoMFormer) [10] is presented as the first continual panoptic segmentation algorithm, using adaptive knowledge distillation in order to tackle the forgetting of previously learned classes. However, there is not still a common consensus on how to design and evaluate a CL method for semantic segmentation, given that it is an open and interesting research area of CV with multiple possibilities of achieving the same objective.

Therefore, the aim of this thesis is to develop a method for learning the semantics of segmented objects in robotic applications by combining video object segmentation methods and CL methods, with a focus on its possible implications in walking or land robots. To accomplish this task, the proposed method will be evaluated in a real robotic scenario using the TORO robot in Deutsches Zentrum für Luft- und Raunmfahrt (DLR) [19], which incorporates a couple camera sensors. The camera images will be processed to detect unknown objects in a given scene and classify these into known or unknown categories. Thus, the method will be validated based on its capability to learn newly observed objects while preserving previously acquired knowledge as well as in terms of accuracy, velocity and real-time efficiency. Although the thesis focuses on only the ability of the method to retain known objects and incorporate unknown objects into its data base, the object segmentation accuracy is excluded from the scope of this thesis, since we will use for this purpose existing models currently under development.

Consequently, we propose Segment and Learn Semantics (SaLS), a method formed by two independent methods, a segmentation tool and a CL algorithm. The objective is to be able to segment one or more objects in a frame, get their boundaries and pass them onto the classification method. This algorithm has to be able to discern between previously seen, known objects, and new, never-seen objects and learn from the latter so that in future frames they can be identified correctly and the method adds the correct semantic information.

The rest of this thesis is organized as follows. Chapter 2 reviews the literature covering state-of-the-art methods for segmenting objects and CL approaches, and for methods that combine both areas. Chapter 3 describes the development of the process for integrating the two methods into a single framework, their parameters and flow of work, but also introduces the dataset that is used in the experimentation part and how the evaluation of the CL part of the algorithm is done, with the respective metrics. Chapter 4 presents the different experiments that we designed to evaluate and validate the CL method and the whole pipeline, and we discuss the results derived from them, obtained from the piloting of the new process in a real scenario in which the method is used with the TORO robot. Finally, Chapter 5 concludes the thesis by discussing the performance of the proposed method and proposing future research in this area.

2 Segmentation and Continual Learning paradigms

In recent years, robotics has emerged as a pivotal field with profound implications for industries ranging from manufacturing and healthcare to exploration and entertainment. Central to the success of robotic systems is their ability to interact intelligently with their environments, a skill that depends on their understanding of the various objects present in their surroundings. As robots become more integrated into human-centric spaces, the challenge of endowing them with the capability to learn the semantics of previously unknown objects becomes increasingly pertinent.

This chapter delves into the existing literature surrounding the task of segmenting images and learning the semantics of unknown objects in robotics. The discussion centers on the integration of CL algorithms in conjunction with advanced segmentation methods, specifically highlighting the SAM-Track approach. Through an in-depth analysis of previous works, this literature review aims to describe the evolution of methodologies, challenges faced, and achievements made in the pursuit of endowing robots with the capability to autonomously learn and comprehend the semantics of novel objects. By contextualizing the use of CL and segmentation techniques within the broader landscape of robotics research, this review aims to pave the way for a comprehensive understanding of the contributions and potential future directions of this hybrid approach.

2.1 Image Segmentation techniques

In CV, one of the most interesting problems rely on detecting and identifying objects in the environment. This is a trivial task for humans, as we automatically process a pair of stereo images of the scene to distinguish the limits between objects, and how far or close they are thanks to a estimation of the depth. However, this task is not that simple for a robotic system, so the development of image segmentation [38, 46] algorithms is needed. This process is in charge of dividing the image in segments, each one would be set of pixels with similar characteristics that form an object. In the literature, there are three main groups of image segmentation: semantic, instance and panoptic segmentation.

Semantic segmentation verses on assigning an object class to every pixel in the image. Instance segmentation just focuses on separating each object in the scene, regardless of which class they belong to. Lastly, panoptic segmentation is a mixture between the previous groups, so it identifies the class of each pixel but separates the different instances of the same class [38].

Due to the importance of this branch of CV, multiple image segmentation algorithms have been developed [46]. Some of the first ones range from thresholding to clustering. In the recent years, this task has relied more on Deep Learning (DL) due to its exponential growth, which has led to a new generation of methods with promising improvements in their performance. Some of these algorithms will be explored in the coming subsections, following as a guideline the work carried out in DLR.

2.1.1 Instance Stereo Transformer

Some of the current approaches demand prior information for each instance under consideration, so there is the need of methods that without any previous knowledge can infer new instances. For example, DLR introduced Instance Stereo Transformer (INSTR) [18] for unknown instance segmentation. This method is trained on synthetic images and is able to extract disparity and RGB-based features to predict unknown objects in the place without further processing, just with a pair of stereo images.

The architecture of the algorithm is shown in Figure 1 and it follows the next stages. The input image to the method is then passed to a feature extraction encoder, which returns a correlated feature representation of both stereo images. These features are processed afterwards by a transformer encoder and a transformer decoder, with the objective of separating the instances. Moreover, a disparity decoder is added in order to predict auxiliary disparities, that is, to obtain a rough depth estimation.

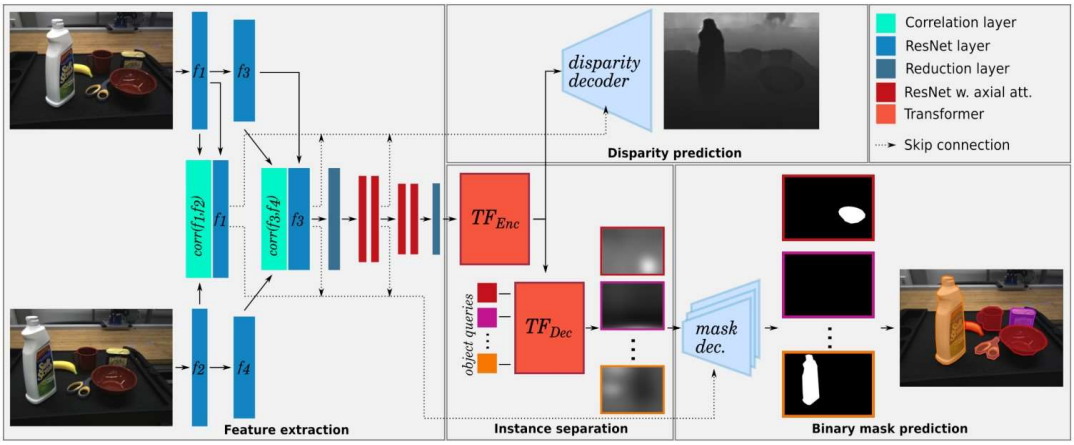


Figure 1: INSTR architecture: after the feature extraction of the pair of stereo images, the instances are separated and the binary masks are returned. Also, INSTR returns a prediction of the disparity thanks to the stereo images [18].

The results of the model applied in real robotics systems show a promising performance, displaying a class-agnostic instance segmentation method able to run at 18 frames per second and segment different shapes and sizes of objects in various backgrounds. In order to fine-tune the method and aim it to one specific task, they extended INSTR and trained it with objects belonging to one class, in this case rocks [7]. In this sense, the ability of INSTR to segment rocks in a scene increased and a positive effect of training on data similar to the test data was shown.

INSTR method constitutes a very strong segmentation method but it is still relying on stereo images. This makes the segmentation task more demanding, as sometimes there is only one single image from the scene. Another drawback of this method is that it has to be trained for the specific application with similar datasets, either synthetic or real, to maximize the accuracy of the predictions.

2.1.2 Segment Anything Model

Meta AI has recently launched a new foundation model called Segment Anything Model (SAM) [31]. A foundation model [8] is a model based on deep NN and trained using self-supervision and transfer learning on a very wide quantity of data at scale. In consequence, it can be adapted to many different tasks and be the foundation for multiple applications different from those seen during training, so it has become a turning point for ML algorithms.

SAM is then a paradigm of a ground-breaking image segmentation method and has became available to the public in April, 2023 [31]. The model overview is displayed in Figure 2 and has three main components:

- Image encoder: pre-trained Vision Transformer that runs every input image.
- Prompt encoder: handles both sparse and dense prompts.
- Mask decoder: returns a mask using the image, prompt and output token embeddings.

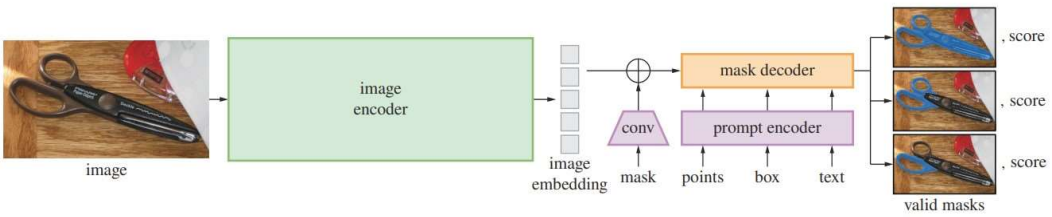


Figure 2: SAM overview [31]. The image encoder returns the image embedding to produce the predicted masks, that can be multiple for the same prompt.

With this setting, SAM is able to predict multiple masks for the same instance and rank them according to an estimated metric that assess how good each one is. It works in two ways, one fully automatic and other accepting input prompts to segment specific areas or objects. The former analyzes a grid of points overlaid in the scene and returns all the predicted masks, segmenting the whole scene at once. The latter takes as input points, boxes or text, and segments only the desired instance. It can return multiple masks with different metrics measurements from the same prompt, so the user can rank and choose the best fit or work with all the possibilities.

However, it also present some limitations. It can generate imperfect masks, missing some structures or producing irregular boundaries, so domain-specific tools are expected to outperform the model. The main objective of SAM is to be a general model, so it is not focused on reaching great scores in segmentation metrics. Moreover, SAM performance is aimed to segment instances, but it does not implement prompts with semantic or panoptic segmentation so it cannot accomplish semantic-level understanding. Further, the most relevant drawback of SAM is that the performance of the model is not in real-time. The prompt encoder and the mask decoder can be optimized and run in a CPU in short time, but the heavy image encoder halts the

segmentation process. This might be the most limiting fact of SAM in some robotic applications, as it is often needed not only a strong accuracy in the segmentation but also a fast pipeline to process the frames in real-time. Along the same lines, SAM does not take into consideration the temporal coherence among frames, meaning that the same object in different frames would not be taken as the same instance.

2.1.3 Decoupling Features in Hierarchical Propagation

Regarding the temporal coherence issue, there are some state-of-the-art projects focused on Video Object Segmentation (VOS), which aims at identifying and segmenting objects of interest in a video instead of a single image. This opens up the possibility to segment objects in real-time, processing each frame from a camera and returning the desired instance masks. One of the most recent methods is Associating Objects with Transformers (AOT) [62], which uses transformers to apply hierarchical propagation for VOS, so that information between frames can be transferred. This method has shown a great performance and can be used in be used in a range of capabilities.

Based on AOT, Decoupling Features in Hierarchical Propagation (DeAOT) [63] utilizes a similar approach but it decouples the features in different branches. This method also proposes a new more efficient module for the hierarchical propagation to alleviate the adding complexity and computation from the different branches. The experiments carried out with DeAOT demonstrate that it achieves a state-of-the-art performance in less time than its predecessors in VOS.

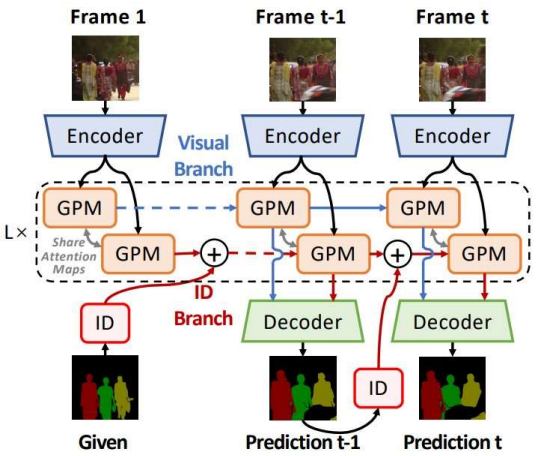


Figure 3: Overview of DeAOT architecture: each encoder decouples the propagation of visual and ID embeddings in two branches, which are then propagated using a GPM. [63].

The overview of DeAOT is shown in Figure 3. The method is able to decouple in two branches both the propagation of visual embedding and IDentification embedding. Then, these branches are propagated using a Gate Propagation Module, which is designed using a single-head attention, more efficient than other blocks used in AOT.

2.1.4 Segment and Track Anything

With the advancement in image segmentation proposed by SAM, this project incorporates the powerful tool so that it obtains the segments of keyframes as the reference for initializing DeAOT. The open-source project called Segment and Track Anything (SAM-Track) [11] uses SAM for automatically segmenting the frames and DeAOT for multi-object tracking of the segmented instances. Since the project is also aimed to develop a unified framework for VOS able to work in different domains, they have integrated Grounding-DINO [35], which effectively fuses semantic with vision modalities and improves the natural language understanding of SAM-Track.

Grouping all these previous features, SAM-Track pipeline is built as shown in Figure 4. There are two operating modes: interactive tracking and automatic tracking, in regards on how the segmentation masks are obtained using SAM and Grounding-DINO. The interactive mode is used only in the first frame and can track an specific object given a human input, such as a point, a box, or a text. Once SAM has predicted the mask of such prompt, DeAOT propagates visual features from past frames to the current frame for tracking. Grounding-DINO acts in this case by selecting interactively the desired objects in the reference frame with natural language commands.

On the other hand, the automatic mode runs SAM every other frame, so that all of the objects present in that moment are segmented and added to the tracking process, and it is able to handle new objects appearing in the video. Once SAM predicts all the masks, DeAOT tracks the new and original objects using a unique assigned ID to each one. This is possible thanks to comparing the tracking results from DeAOT and the annotation results from SAM, so that the method is able to distinguish between already seen and not seen objects. This is achieved with the Comparing Mask Results (CMR) block, that returns either 0 or 1, if there is or not temporal coherence.

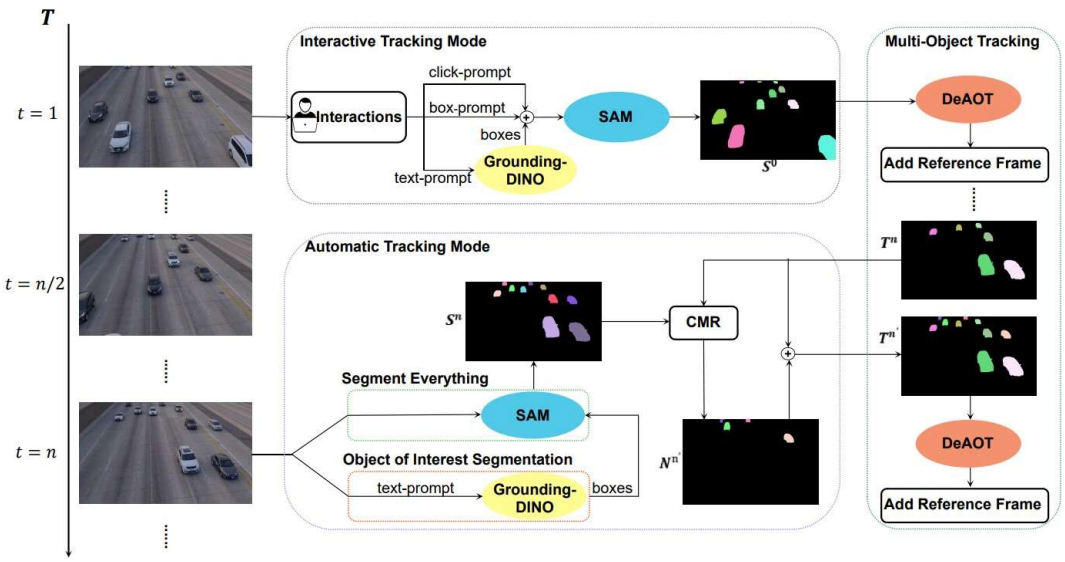


Figure 4: SAM-Track overview [11]. There are two modes: interactive or automatic, SAM returns the mask annotations in both cases and the tracking is updated.

The performance of the method has been validated with different experiments using popular VOS benchmarks and they have shown the efficiency and robustness of SAM-Track in different applications, ranging from autonomous driving to medical field settings.

2.2 Continual Learning algorithms

In the recent years, ML models have been increasing their performance, achieving even human levels. However, these models are static and cannot adapt over time to new data, so there is the need of systems that can continually learn over time in the same way as human cognition learns concepts. It is also possible for us to revisit old concepts and some of the information might be gradually lost, but it is not common to totally forget old knowledge. This is an issue in ML, as the networks suffer from what is called catastrophic forgetting [13, 36]: they forget old concepts as they learn new ones, so they are unable to perform well on previously seen data after updating with recent data. This issue is a direct result of the stability-plasticity dilemma, a general problem in ML, in which stability refers to the ability to retain previous knowledge and plasticity to the ability of integrating new knowledge.

For this reason, CL [13, 36] – also referred to as lifelong learning, incremental learning or sequential learning – is the part of ML that studies the problem of learning from a virtually infinite, independent and identically distributed stream of data, with the aim of preserving and gradually broadening acquired knowledge. This stream of data can be very diverse, having data from different input domains or associated with different tasks, but also with only a small portion of input data available at once.

2.2.1 Families of CL methods.

Even being a fairly novel concept, over the past years a wide range of methods have been developed to solve CL issues and some real-world applications are beginning to emerge. There are different classifications of these algorithms, albeit one of the most clarifying ones is presented in Figure 5. Therefore, three main families are identified [13] in regards on how they store the information for each task and use them for learning sequentially, and some example methods from each group are displayed:

- **Replay methods**: samples from previous tasks or generated pseudo-samples are replayed while learning a new task, either to re-learn or to prevent task interference. There are rehearsal or pseudo-rehearsal methods that re-train the model with a subset of stored samples or with the output of previous models, in each case. Constrained optimization algorithms belong to this group too and they try to avoid overfitting by restricting interference between new and previous tasks.

- **Regularization-based methods**: a extra regularization function is introduced in the loss, reducing memory requirements. They can be either data-focused or prior-focused methods, depending on whether they use knowledge distillation

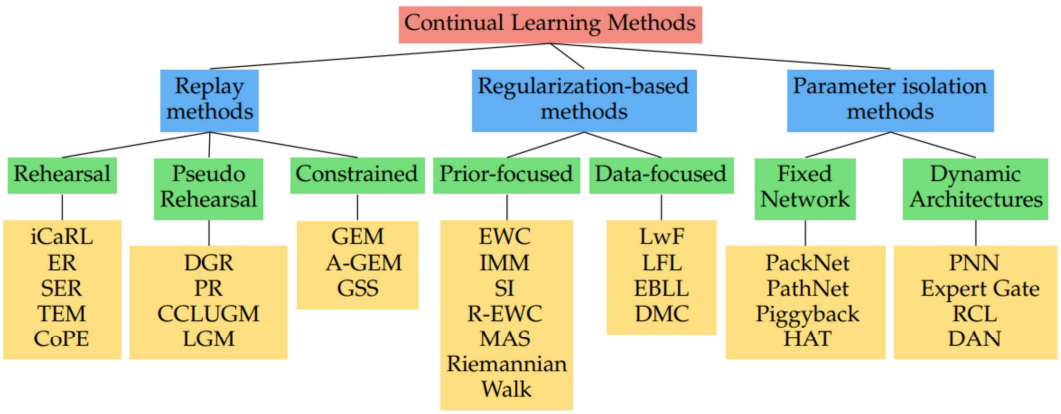


Figure 5: Diagram displaying the CL families of methods with examples [13].

from previous models to the new model, or use a prior that is a estimated distribution over the model parameters when learning from new data.

- **Parameter isolation methods**: different model parameters or even dedicated model copies are addressed to each task. They are mainly differentiated in fixed or dynamic architectures, if they do not change their architecture when training with new data, or if they do grow new branches.

All three families have advantages and disadvantages, and using one algorithm or another mainly depends on the preference of the user and the nature of the application. For instance, both replay and parameter isolation methods might have bigger memory consumption than regularization methods, as they are hoarding information for each different task. On the other hand, regularization methods comprise additional loss terms for consolidation previous knowledge, which might result in a trade-off on the performance of former and new tasks.

2.2.2 Replay methods: Random Forest

Multiple methods and variations of CL algorithms can be found in the literature. Tracking the research in DLR, they developed a method that falls in the replay category in order to continuously learn objects from new unseen classes [15]. It is based on a Random Forest classifier [9], a very efficient framework consisting in a number of binary random decision trees. To train each one of them, the data is repeatedly divided in two subsets along a feature dimension according to a threshold, until a stop criterion is met. When doing the inference, the class is determined according to the most frequent leaf node that the sample falls into among all of the decision trees. Consequently, Random Forest is an algorithm with a low variance and risk of overfitting, giving that is formed by multiple weak classifiers. This work also included three improvements to deal with the persistence of knowledge, the efficiency of the memory usage, and the interruptibility of the learning process.

To achieve the first characteristic, they replaced old trees with new, trained trees to add unseen objects. Moreover, in order to save samples from previous tasks, they added

a pool setting in which a fixed container is filled with the same number of samples for each class that appear, while a flexible container has a varying size depending on the classification error of the corresponding category. With this setting they classified different objects in a task incremental way, that is, training with data of one task at at time until convergence. The modified Random Forests appear to be a promising method in a CL environment, with a classification accuracy similar to most of the state-of-the-art methods, so that it can be used in many robotic applications.

2.2.3 Parameter isolation methods: Progressive Neural Networks

In spite of the different variety of developed algorithms, there is one that has been more present in some of the DLR most recent works. This approach is a parametric isolation method, PNN [57], an architecture that takes advantage of an existing network to transfer knowledge between tasks. The aim of the project was to learn to classify new objects by adapting a trained classifier and avoiding fine-tuning, which was the most used method in these cases. Therefore, the adaptation of the architecture is done directly on the target network, totally independent from the source network, allowing for flexibility in size and structure. Another key component of the PNN is the lateral connection from the source network to the target network, used to derive useful information from previous data with the objective to learn a new task. This fact enables to transfer learning, which is the ability to accumulate and transfer knowledge to new domains, something natural for humans but complex to integrate in a machine.

For this work, the chosen base network was a Convolutional Neural Network which requires less computation to train and less memory consumption, at the expenses of less accuracy and simpler models. Apart from testing the original setting for comparison, they also applied a series of modifications on the PNN so that the network flexibility was increased, and just a few parameters in the internal layers were changed. Both methods were then evaluated in four different object recognition datasets, and the proposed PNN outperformed the original design in every experiment, both in transfer learning approaches and extending a classifier with new classes, with the advantage of the accuracy not being influenced by the amount of available training data. This is specially beneficial in those cases where there is an unbalanced distribution of classes so that the new objects are often under-represented.

The PNN has also been explored in other context different from CL, for instance in state-of-the-art researches such as the one carried out by Google [48]. In this case, the objective was to apply CL to solve complex tasks on a robot as the current Reinforcement Learning approaches are very slow for real-time applications. The proposed solution was to transfer learning methods in order to bridge the reality gap that separates simulations from real world domains using PNN, as they have been proven to have notable performance in this field. This was motivated by three main reasons: features learnt for one task might be transferred to new tasks without destruction; columns might be heterogeneous and able to solve different kinds of tasks; and they add new capacity when transferring to new tasks.

The implementation of the PNN in this setting started with a single column (or network) with layers trained to convergence. When switching to another task, the

parameters were frozen and a new column was initialized receiving inputs from layers on the first network via lateral connections. Each one of the columns was trained to solve a particular problem related to robot control and this configuration could be also modified to accommodate for different task difficulties for instance. The experiments had three phases: first they trained the simulation, secondly they transferred the knowledge to the robot, and finally they transferred the knowledge to a dynamic robot task with proprioception. The experiments focused on the task of reaching a visual target, with rewards provided as feedback. This required that the state of the arm and the position of the target are correctly inferred from visual observations, so that the robot learns robust control in a high-dimensional state space. At the end, they were able to prove that PNN can be used to achieve reliable and fast transfer for pixel-to-action robot control policies.

2.3 Adding semantics and Learning Online

The segmentation methods explained in the previous Section 2.1 allow to recognize and track one or multiple objects in a image or across different frames of a video. However, there is still not a clear panoptic perception as the methods cannot identify the category of the object. In a robotic system the goal is to detect nearby objects, but also to know how to interact with them so the type of object is essential for this purpose. The first step is then to segment an object and categorize it, that is, add the semantics to the segmentation. Not only is important to classify a given instance, but also to detect if it is in the data base or not, so these are the two main keys for the panoptic perception in a vision system.

Once this is achieved, the next goal is to learn from the unknown. The synergy between CL and image segmentation holds immense potential for enhancing the perception and decision-making capabilities of robots, allowing them to autonomously understand and interact with complex environments. This section delves into the dynamic realm of integrating the principles of image classification with CL, which enables models to adapt and learn from continuously incoming and previously unknown data, but also with advanced image segmentation methods, which are pivotal for scene understanding and object recognition, so that the robots are empowered with more context-aware perception abilities. The following works exemplify the collaborations between these domains, highlighting their contributions to the advancement of robotic systems.

2.3.1 INSTR with semantic information

Following the line of work in DLR, INSTR is a solid segmentation method great performance but it is missing the semantic information. The next focus is to apply it in real-world robotic systems in which it is utterly important to detect both known and unknown objects in order to navigate, manipulate items and detect possible anomalies in the environment. In this way, they implemented two modifications to INSTR so that it could reason about semantics in the scene but also separate identified instances into known or unknown categories [6]. To address the first implementation, they

aimed to have a panoptic segmentation so that a label would be assigned to each pixel while detecting each single instance. To achieve so, a segmentation head was added to INSTR and the output was fused with the instance predictions.

Regarding the second extension, they followed previous works based in Out-of-Distribution (OOD) detection [61], a term that has gained increasing attention and has led to multiple developed methods with different approaches. It is based on the nature of the training data and testing data, whether they are drawn from the same distribution or not. In a closed-world assumption, both sets are supposed to belong to the same distribution but that is not the case in an open-world setting, as there might appear unexpected OOD samples with semantic – different classes –, or covariate – different domain – shifts. This problem is often related to other similar issues such as anomaly detection or open set recognition, very close in methodology and goals.

The method to solve the OOD samples used in INSTR is a simple Bayesian Neural Network, defined as a stochastic artificial NN trained using Bayesian inference [29]. They explored two different settings with different methods including Dropout, which relies on preventing overfitting by dropping out neurons during training [3, 23]. The main issue is that the dropout probability has to be correctly calibrated which can become computationally expensive.

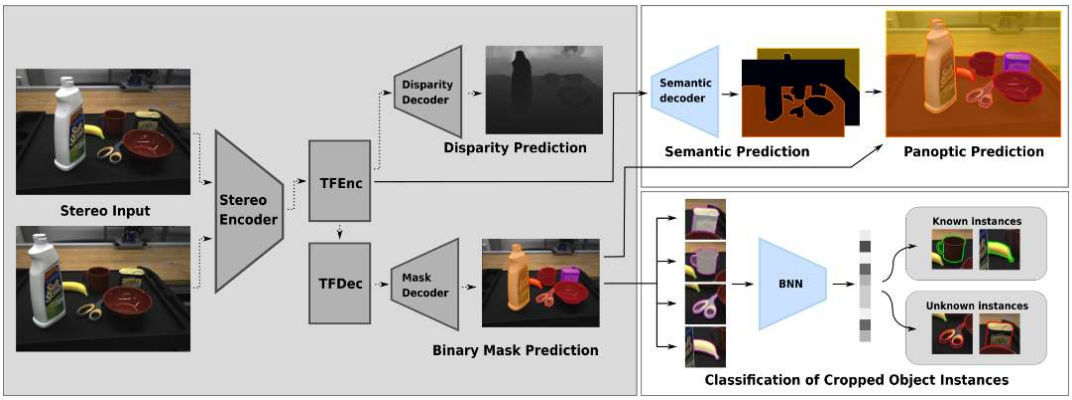


Figure 6: Overview of the INSTR extensions [6]. INSTR pipeline has a gray background, the panoptic segmentation is in the top right, and the separation between known and unknown instances is in the bottom right.

The whole framework is shown in Figure 6, where both INSTR and the extensions pipelines are displayed. The experiments validated the panoptic segmentation and the differentiation between known and unknown objects, showing that the framework is robust. As the main focus of this project was its application to robotics, they also did a experiment for grasping stones and for table clearing. The quantitative and qualitative results indicate that the presented extensions can correctly be used for these purposes efficiently.

Nevertheless, this work does not actually apply any CL algorithm, but it adds semantics as the first step towards a complete understanding of the surroundings of the robot. They achieve the identification of objects whose label is not in the dataset, considering it as unknown. In this way, a robotic system encountering an unknown

object and identifying it as such is a great progress in the field, but most of the times the robot will avoid interacting with it due to said lack of knowledge.

2.3.2 Lifelong Robotic Vision competition

For the International Conference on Intelligent Robots and Systems (IROS) in 2019 there was held a Lifelong Robotic Vision Competition focused on Object Recognition [52]. It was motivated by the concept of lifelong learning, to understand the current world using previous knowledge autonomously. This process, simple for humans but very complex to transfer into robots, has been fostered thanks to the recent advances in CV and DL methods, but robotic vision has unique challenges given that existing datasets assume a fixed and closed environment with time-invariant task distributions. This competition aimed to analyze the use of knowledge from previous tasks for learning new tasks, in a efficient way. The goal was to test the capabilities of different CL algorithms in a service robot scenario, and over 150 participant teams competed for that purpose.

The challenge dataset was first presented in the competition, the OpenLORIS-Object dataset [51, 52], with the objective of providing a general benchmark for lifelong learning methods with everyday objects in different types of illumination, object occlusion, object size, distance or angle between object and camera, and clutter information. The data was collected using a real robot with multiple sensors for RGB-D information, and it was formed by 17-s videos with 260 chosen frames for each one of the 69 instances under 19 categories. These objects were considered under four different environmental factors (home, office, campus, and mall) and three environmental difficulty levels. In total, they provided 430 560 images for the challenge with bounding boxes and masks. Therefore, the focus for the participants was not on the segmentation of the objects in the images, but on keeping the knowledge and learning new tasks. The evaluation metrics considered were not only focused on the overall accuracy of all the tasks, but also on the model efficiency (model size, cost, and replay size).

From all the participants, only 8 were in the final phase of the competition. They proposed DL models with different strategies: mainly regularization methods (knowledge distillation), and few parameter isolation methods. Although the main objective of the challenge was the CL algorithm, some of the teams also explored other CV methods in order to improve the final solution, such as Single Shot multi-box Detection or data augmentation. The winner team employed a dynamic NN, shown in Figure 7, with two parts: expansion for data across different domains and knowledge distillation for similar domain data. The domain gap was computed using previous models, so they were able to detect known instances in the current task without using previous data. Then, these known samples were used for knowledge distillation, employing the best head for that purpose. This implementation gave a high accuracy of 96.86%, not the highest of all teams but it was accompanied by a small model size of 16.3 MB; one of the smallest inference time of 25.42 seconds: no replay size given that their method did not store samples from previous tasks; and the highest accuracy in a more challenging bonus set. It is worth noting that none of the methods presented

in the challenge were aimed for a real-time application, as the one with the shortest inference time was of 22.41 seconds. However, it was not a requirement, although the inference time was part of the final mark for each algorithm, being the accuracy the most important metric.

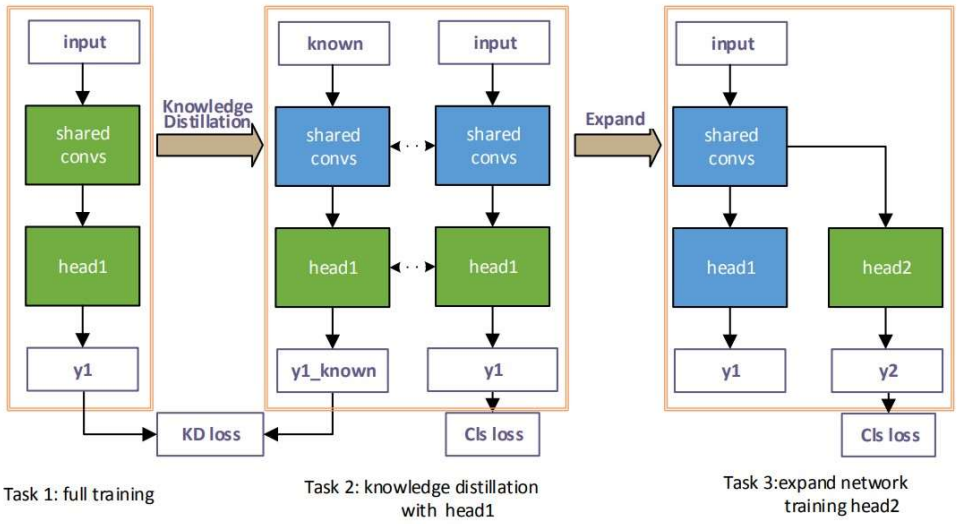


Figure 7: Architecture of the winning algorithm in IROS 2019 [52, 65], describing a dynamic NN.

This competition focused on the CL part of this thesis, but it used a new dataset collected with a robot for training and testing. In this sense, it stated the relevancy of the subject in hand, as thanks to the recent advances in both CV and CL we are reaching the integration of both disciplines in one single pipeline, very necessary for the future of robotics in order to adapt to new environments. Moreover, the legacy of the OpenLORIS-Object dataset might signify a starting point for comparing CL methods using the same benchmark, while orienting the algorithms to CV applications in robotics.

2.3.3 Current Continual Semantic Segmentation paradigm

In these recent years there have been multiple approaches to CV tasks in a CL setting, both focused in image semantic segmentation but also in VOS. One of these works is Pseudo-label and Local Pooled Outputs Distillation (PLOP) [17], which aims to solve both the catastrophic forgetting and the semantic shift of the background class, i.e., when the pixels taken as background contain pixels belonging to old and unseen classes, which might foster even more the catastrophic forgetting if they are not identified correctly. To solve these issues, they propose a spatial distillation scheme, that helps to retain the knowledge better by keeping spatial relationships, and a pseudo-labeling method to detect old class pixels from the background. This method was then validated using three different datasets: Pascal-VOC 2012 [20] with 20 classes, ADE20K [66] with 150 classes, and CityScapes [12] with 19 classes; and displayed an improved performance over other state-of-the-art methods at that

time. This approach integrates image segmentation and CL in one single method that performs both duties indistinguishably, and takes care of the main challenges in these kind of algorithms.

Later on, another work explored the replay approach in CL for image segmentation, in Replay in Continual Learning (RECALL) [37]. Previous works, as seen in [52], rely more on knowledge distillation for image classification, but they often fail when multiple incremental steps are performed, or when the forementioned background shift occurs. As a response to this problem, they propose a replay strategy to generate samples from old classes by re-creating representations and mixing them with objects belonging to new classes during the training process. To achieve this objective, they propose two ways: using a pre-trained generative model that produces samples of a given input class, or an approach based on online crawling images from the web, so that they can extract useful keys from weakly supervised samples and use them to generate pseudo-labels. Meanwhile, to alleviate the background shift they use background inpainting, that is, they take the background of each ground truth map and label it with the prediction from a previous model. This approach is very interesting because they use a new unexplored technique in this field, which is looking for data on the web. However, it also comes with a lots of limitations and question marks, but their initial testings in the Pascal-VOC 2012 [20] dataset were very positive, with room for improvements in how to control the weak supervision during web-crawling.

In order to compare different approaches and see patterns in the development of CL and image segmentation, a work in CL for class and domain incremental semantic segmentation [30] explored some of the already existing solutions and analyzed them. For that purpose they used two available datasets, CityScapes [12] and BDD100k [64].They discovered that most of the times, those algorithms based on knowledge distillation were able to perform better in class-incremental settings, i.e., when each task introduces objects belonging to new unseen classes, as it had problems with adapting to new data; but replay methods were more useful when the learning is done in a domain-incremental fashion, that is, when there are objects of the same class but the input distribution (different context) is changed.

These past works are more focused on semantic segmentation, rather than in panoptic segmentation. In a real-world setting, it is important both to assign a class label to each segmented object in a frame, but also to be able to distinguish different instances belonging to a same class. For this reason, Continual MaskFormer (CoMFormer) [10] is presented as the first CL model able to operate in both semantic and panoptic worlds. They explore the recent transformer approaches in order to treat the segmentation task as a mask classification issue. In this way, they propose an adaptive distillation loss accompanied by a mask-based pseudo-labeling method, and they surpassed the performance of other methods such as PLOP on the commonly known ADE20K [66] dataset.

Continuing in the panoptic segmentation line, a recent work called Continual Learning for Depth Estimation and Panoptic Segmentation (CoDEPS) [56] explores the panoptic side while adding a depth estimation task. It is based on experience replay with a pseudo-labels generation but limiting the size of the replay buffer so that it can be implemented in mobile robotic systems with scarce resources. The method is

able to process an online stream of images from a camera, which are combined with samples from the replay buffer containing annotated images and previous samples. This technique allows pseudo-supervision on the target and an additional network is used for unsupervised depth estimation. This is one of the few algorithms that explicitly states the possibility of working online with a stream of images, as it was conceived with the idea of autonomous cars in urban scenarios, turning it into a VOS method for panoptic segmentation and depth estimation. It was evaluated in different datasets from a variety of domains, such as CityScapes [12], KITTI-360 [34], and SemKITTI-DVPS [4], the last two being recently released for the domain of autonomous driving and containing LiDAR data apart from 2D and 3D panoptic annotations.

Another method that clearly states the online VOS is the one proposed in [39]. This work develops a regularization-based CL approach based on an existing baseline, Learning What to Learn (LwL)[5], a popular VOS few-shot learner approach. Moreover, they introduce a new dataset for CL in VOS, called CLVO23, with long-video object segmentation as a more realistic and challenging dataset in this field. The VOS method is thought the be added to any existing online framework, to improve its efficiency and resistance in distribution shifts but keeping its accuracy. However, the resulting performance was not better in short video datasets, such as DAVIS16 or DAVIS17 [45], where the perspective of the object does not suddenly change.

Method	**Approach**	**Publication date**
PLOP [17]	Local POD + pseudo-labelling	Mar 2021
RECALL [37]	Replay data + background inpainting	Sep 2021
CoMFormer [10]	Adaptive distillation loss + pseudo-labelling	Nov 2022
CoDEPS [56]	Replay buffer + pseudo-labelling	May 2023
LwL-based [39]	Regularization-based	Apr 2023

Table 1: Summary of relevant state-of-the-art works on continual segmentation.

All of this works have been developed in the last three years, so it is understandable that there are multiple different approaches and we cannot really discern a clear direction in the integration of CL and VOS. As most, we can establish that the current trend for image segmentation is more focused in the use of vision transformers, while for the CL setting, we encounter methods belonging to the three families: replay, regularization, and parameter isolation; and most often, features of different types combined. Knowledge distillation, in the data-focused subgroup, is the preferred algorithm for transfer and continual learning among the others in the regularization-based methods, although it is sensitive to the domain shift problem. In addition, even though there are some datasets whose use is more extended in the CL area, including CityScapes or Pascal-VOC 2012, they do not fulfill the current needs in the field, so there are some others, such as CLVO23, that are starting to appear as powerful

benchmarks which might mark a more constant baseline for the development of new algorithms. All in all, we have multiple options to choose from in order to carry out an online VOS with CL, and the decision of which algorithm to use heavily depends on the main goal of the application, but multiple methods might be suitable for the same purpose. The main works have been summarized in Table 1, in order to give an overview of the current CL segmentation panorama.

2.4 Future applications in Robotics

In the previous Sections 2.1 and 2.2 the main algorithms and works related to image segmentation and CL have been described. The integration of both fields in a single pipeline has been explored in Section 2.3, but the objective of this thesis is also to apply it in a real robotic application. Therefore, there are different projects in DLR that might benefit from a pipeline that detects instances in the environment, classifies them, and learns from the unknown objects. In this Section 2.4, they will be described in order to illustrate with examples how the method in this thesis could fit and extend the capabilities of the existing robotic systems with very interesting applications.

2.4.1 Robotics for planetary exploration

One space-related application that might need a complete segmentation of the scene in an autonomous and precise manner is the planetary exploration setting. There are multiple active and developing missions present in the different space agencies aiming for a mobile robotic systems able to semi-independently explore new terrains in extra-terrestrial bodies, being some of the most known the martian rovers, Curiosity [55] and Perseverance [21], or the lunar rovers, Yutu [32] and Yutu-2 [33]. The main issue in these cases is that rovers cannot be remotely controlled if the distance between the mission control station and the rover itself is very big, as the radio signals will take longer to travel it. For instance, just a communication between Mars and Earth can take up to 20 minutes, so there is the need for the rover to operate autonomously in terms of navigating or acquiring data. Even if nowadays is common to also have some human inputs for critical decisions in the mission, the goal would be to fully automatize the process. If a rover is sent to a more distant body, such as Titan, it will become even more difficult to have contact with it in order to take navigation decisions, so there should be implemented an algorithm that is not only able to analyze the current environment but also learn from it to take the most accurate decision in the future.

Apart from the navigation, most of the times these mobile robots are equipped with instruments or cameras that might also need to be operated, introducing another level of complexity to the system. Furthermore, as seen in the Mars 2020 mission [2, 21], we will be seeing not only mobile robots moving through the terrain, but also other types of robots that are connected between them and are able to interact to a certain extent with each other. Perseverance is accompanied with a small helicopter, Ingenuity, that is a technology demonstration of the first autonomous controlled flight in the red planet. Apart from its main purpose, it could help in other roles such as

working with the lander to provide terrain sampling and investigations. The base station of the helicopter is located on the rover, which identifies it as a science payload, and serves as interface between both robots.

Even though Ingenuity is just the first helicopter of its kind to be involved in a mission in Mars, future missions could deploy more of these helicopters in order to execute a wide range of scientific missions that would complement the rovers and landers discoveries. Consequentially, some planetary exploration researches are focused on teams of heterogeneous and mobile robots that operate on extra-terrestrial bodies co-dependently, with an improved efficiency due to the parallelization and redundancy of tasks.

DLR has participated in a state-of-the-art project by the Helmholtz Association, with the aim of developing a robotic system that is heterogeneous, interlinked and autonomous. The ARCHES project [50] is aimed to solve these challenges, so that the robots have to rely in sensors to explore unknown areas in a independent way. The project was finished in 2022 with a demonstration mission in Mount Etna [58, 59], and it marked the starting point for future missions in different applications ranging from ocean to space exploration. In this latter topic, ARCHES will serve as a demonstration to validate relevant scenarios and technology that will surely be useful in the next years, with the returning of humans to the lunar surface, the deployment of the Lunar Gateway, or the foreseen manned missions to Mars.

The robotic system is formed by different components, and each one of them require to be completely autonomous in order to execute coordinated actions with the other robots, so that they need to compute their own location online, identify their surrounding environment and share the collected data. The team, shown in Figure 8, is constituted by:

Figure 8: [58]

- **Dron ARDEA**: aerial hexacopter that serves as a scout that can explore areas with difficult access for the land rovers, and has multiple autonomous skills for that purpose.

- **Scout Rover**: agile robotic system with rimless wheels that enable it to roam in different complex terrains that other rovers might not be able to access.
- **Interact Rover**: four-by-four wheeled rover with a camera and arms aimed for proximity actuation to collect in-site samples.
- **LRU1**: mobile land vehicle able to analyze the terrains closely with the science camera, and serve as a transport to ARDEA.
- **LRU2**: second land rover with a manipulator arm attached that enables it to grab samples and transport payloads, apart from housing scientific instruments.
- **Rodin Lander**: stationary unit that functions as a base station and communication node, while defining a coordinate frame for all the subsystems.

The main objectives of the demonstration mission in Mt. Etna were two: exploring and sampling geological interests in the area and deploying a Low-Frequency Radio Array working as a telescope. The planning of the mission started with an initial phase called GEO I, in which the robotic systems were teleoperated, following by the second phase GEO II, in which geological samples were collected and the site was explored. The last phase corresponded to the deployment of the telescope and a permanent base installation.

In GEO I, LRU1 and LRU2 rovers worked for around three hours in a semi-autonomous manner to collect the data close to the Lander. They navigated and manipulated objects during all that period, and Mission Control was in charge of sharing with the robots the skills to be executed and receiving the data. LRU1 explored three different difficult-access locations in order to scan them with the cameras and LRU2 was in charge of picking up sample boxes with its arm and navigating to sampling sites to pick up rocks or soil of interest. In order to identify the relevant samples from others, INSTR [6] was employed for stone segmentation, to provide masks for further selection or grasps them autonomously.

In GEO II, the Interact and Scout rovers explored the area close to the Lander and collected samples operated by an astronaut. In this phase, it was successfully demonstrated the cooperation between the different component of the robotic system.

All of the demonstration phases were successful and the ARCHES mission was proven to be efficient, cooperated and very autonomous. The planned experiments were fully achieved, although some of the current technologies and methods might need optimisation in particular fields, the joint and cross-domain use of the robots led to strong results that promote the use of this type of systems in future unmanned missions.

As it was mentioned before, INSTR was used as a segmentation tool in the mission in order to detect relevant samples in the surface. However, the method developed in this thesis can help in a major way, almost all of the robots in the mission were equipped with cameras so they would surely benefit from the detection and classification of objects but also from the CL setting, as they could encounter objects never seen before. Consequently, they would be able to operate in an almost fully autonomous way, as the

few times that the robots had to be operated in the mission were when it was needed a better understatement of their surroundings. However, if the robots are the ones that do so, there is no need for human intervention. We think that this is a niche field for the method proposed in this thesis and it is very relevant for future missions, as the ones mentioned to the Moon or Mars.

2.4.2 TORO walking robot

Apart from the participation in the ARCHES mission, in DLR there are currently multiple robotic projects being developed, such as Bert [22], Rollin' Justin [42] or TORO [19]. These robots are the paradigm of complex and state-of-the-art systems with multiple applications in the real world, ranging from personal assistance to specific handwork or multigrasp manipulation in hazardous environments, as space or natural disasters. They explore a wide range of mobility problems very relevant in robotics and their continuous development enrich our knowledge in these fields. For this reason, DLR aimed to create a humanoid system that is similar to us in appearance and abilities in order to understand the human body. There have been some predecessors that built bipedal robots in both industrial and academic contexts but the most relevant issue is related to the control of each individual joint, so there has been an accused incise in learning the robot dynamics for torque controlling. In DLR, this project started with a biped system that was gradually developed to the full torque-controlled humanoid robot TORO [19].

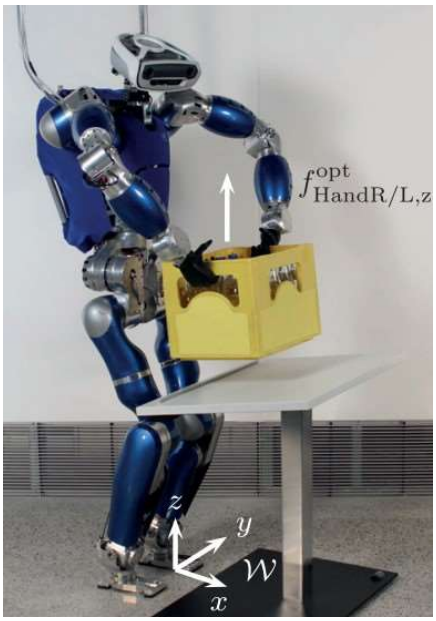


Figure 9: TORO picking up a box while maintaining balance [28].

The main objectives with TORO were focused on multi-contact balancing, human interaction and manipulation tasks. In Figure 9, TORO is shown lifting a box with its two arms, one complex interaction that it is able to perform standing in its two legs and balancing. For that purpose, previous robots such as the biped and Light

Weight Rovers were reused and most of its parts were assembled in TORO, focusing on evaluating torque based control approaches and improving its own skills. The result is then a human-size robot with a height of 174 cm and a weight of 76.4 kg, able to move thanks to 25 torque-controlled and 2 position-controlled joints. Its feet are relatively small in order to allow full contact with small surfaces, and its arms have human dimensions able to carry 5 kg of weight each.

The head contains the sensors and a computer system to estimate the motion and build a 3D environment map. The data for this purpose is obtained with a pair of stereo cameras processed on a FPGA with a resolution of 0.5 MPixel and a rate of 15 Hz. It is also equipped with a RGB-Depth estimator sensor working at 30 fps that provides information of the objects distance in the range from 0.7 to 4 m. The electronics, including two battery packs that power the robot, are mainly located in the backpack, but the robot can also be powered via a power cable, requiring approximately 250 W in steady state. In this part the two main computers are located, one used for control in real time using Simulink and the other for planning and communication between sensors and drivers. There is also an inertial measurement unit in the thorax to estimate its orientation and acceleration. More information on the control strategies, on how TORO walks and balances in different environments solving various skill problems are detailed in [19]. During the latest years, new control techniques have been approached in order to improve the mobility of the robot, such as the ones explored in [27] or [28]. In this sense, TORO has became a reliable project for research that contributes to the design of the next generation of humanoid robots.

Regarding the purpose of this thesis, the focus is on the Perception and Cognition of the robot, how it senses the vision and tactility in order to be teleoperated or behave autonomously. TORO uses the camera sensors to detect AprilTags [40], a visual system commonly employed in robotics and augmented reality that is very simple to implement. These tags can be printed and the detection software is able to detect them and precisely compute their position and orientation, all of that achieving real-time performance in most of the popular processors.

The system is based on a lexicographic coding system, with a robust software able to withstand different lightning conditions and angles of view. The AprilTags can be compared to the QR Codes in the sense that both are 2D bar codes, but they encode less data in them to be more robustly and accurately detected. For their use in TORO, these AprilTags have been located in different objects in the laboratory, identifying diverse types of terrains in which the robot will step, shown in Figure 10. In the experiments [54], the robot identifies the AprilTag and walks to the predefined location with respect to it. The size of the different walkable objects and how far they are from the tags are hard-coded, so just the position of the AprilTag is needed to start the walking process over the terrains.

This AprilTag system is currently implemented in TORO and used every demonstration walk. It has been proven to be very efficient, as it runs fully on-board and real-time with no lag or stop. Although it is working properly, since the tags are correctly identified and the robot walks accurately to the predefined location, it lacks freedom on the robot movements. That is, the demonstration is executed in the laboratory, a closed environment fully controlled, and the movements of the robot

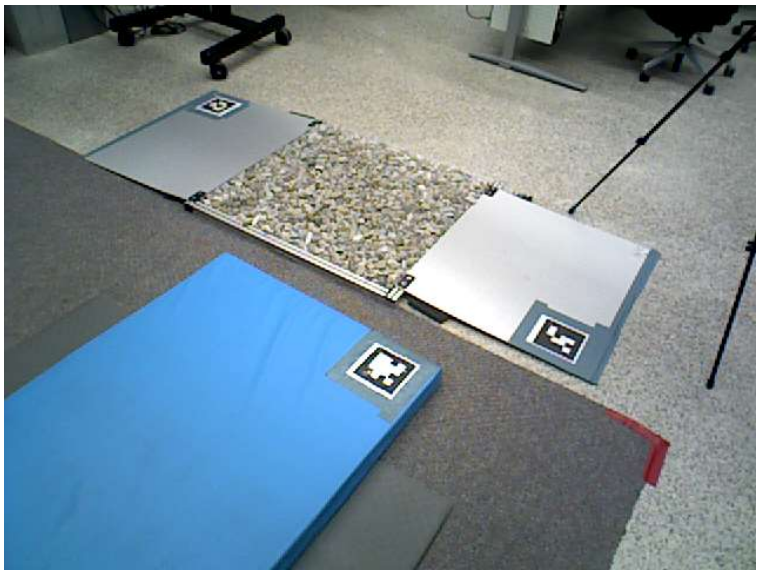


Figure 10: TORO lab with AprilTags identifying some objects. In this setting, the robot detects the tags, walks to the next position and computes how far the object is, so it can calculate where and how the next step is going to be.

are so too. The team working with the robot wants to improve the vision system of TORO so that it does not depend on the AprilTags, because it is as efficient as limiting. Moreover, the software is not very recent and even if it is widely used nowadays, there are tools that have been developed in the past few years that might improve the performance of the vision system in TORO.

Ideally, the robot should be able to detect instances of different terrains correctly, then classify them into known and unknown categories, and ultimately provide depth information with the sensor. A method like the one developed in this thesis might be suitable for this purpose, so the robot would still be able to detect the different terrains and compute how close or far they are in order to prepare the next step, without the need of any external tag marking the objects. It could also detect new unknown objects, for instance, a human in the way of the robot or a new unexpected object. Consequentially, the knowledge of the environment would be much deeper than with the tags, and TORO will be able to move and interact accordingly.

Moreover, it is worth noting that the robot has different walking styles, depending on the kind of terrain. For hard surfaces, such as the floor or the mat, the robot is able to walk faster and in larger steps. However, when walking over a mattress or stones, the steps are much shorter in order to not lose balance, so the speed of the robot is slow. It would also be interesting to integrate these different walking parameters into the vision system, so that depending on the terrain that the robot detects in its vicinity, it changes its way of walking to maximize the performance and the balance of the step.

3 Segment and Learn Semantics method

This chapter delves into the materials and methods employed in the pursuit of advancing video segmentation techniques and facilitating CL through the integration of SAM-Track with PNN. Video segmentation stands as a pivotal challenge in CV, requiring precise identification and classification of objects and regions within video frames. This chapter explains the approach undertaken to refine and augment conventional video segmentation methodologies. Additionally, it outlines the fusion of SAM-Track and PNN as a means to enable dynamic knowledge accumulation in robotic systems.

Traditional video segmentation methods often cannot cope with the complexity and dynamism in videos and real-time transmissions. By using SAM-Track, this research capitalizes on a versatile framework capable of identifying and tracking objects across consecutive frames, fostering a more robust foundation for subsequent analysis. Furthermore, the incorporation of PNN introduces a paradigm shift in the realm of CL. Unlike conventional NN, PNN facilitates the progressive accumulation of knowledge over time, allowing the model to adapt and expand its capabilities..

The following sections provide an in-depth exposition of the research methodology, encompassing the details of SAM-Track and PNN, their integration in a single pipeline, the dataset that will be utilised in the experimentation part and the evaluation procedure in order to validate the performance of the designed method.

3.1 Video Object Segmentation: SAM-Track

The proposed method is aimed to be used in real-time so that the images obtained from a camera are processed almost immediately. With SAM this is not achievable, but SAM-track makes up for it and is able to process the images at considerably high frames per second. Consequently, this is the tool that is going to be used for segmenting the instances in the frames, in a way that the annotations in different frames have temporal coherence. SAM-Track is a very recent tool, so it is understandable that it might still need some improvements, but it is a state-of-the-art method that does not need to be trained, serving as a foundation model for mostly any application.

3.1.1 Automatic and Interactive modes

The method can be utilized in two different ways, and we have adapted them to the purpose of the thesis. An example of frames segmented using both modes is shown in Figure 11. The automatic way works very similarly to the original automatic method. Every `sam_gap` frame, the automatic segmentation is performed. This is done by SAM and it returns a mask identifying with different numbers each one of the different segments of the frame. Then, this results is passed to the tracker, which adds the reference of the objects and is able to follow their temporal evolution through different frames.

Regarding the interactive method, we have only implemented a point input prompt. In this context, we will not need text or box prompts, as we are emulating the human sight. With a point, we can focus the segmentation task to only one object, in a very

similar way as how we see. This prompt can be any point in scene, but we take the centre as the reference, although other possibility could be a lower point closer to the feet. When we are walking, our sight is not strictly focused in the floor but we are still aware of what is below us, the type of the terrain we are walking on. Either ways, the method will only segment the object in which the point is contained. Once it has been segmented, its reference will be added to the tracker. When the method detects that the point prompt is in a new object that has not been previously segmented, the segmentation process will be run again to identify the boundaries of the new instance. This is possible thanks to the tracker, as the limits of the already tracked object are known in every frame and they are temporal coherent.

Although these two possibilities are available, for the purpose of the thesis it is better to have the segmentation focused on only one place, imitating the human vision. Moreover, the point method is much more faster than the automatic one. With the automatic mode the whole image has to be encoded by SAM which takes much longer than segmenting with a given prompt, because it is more limited. Moreover, in a real-time environment any delay can be critic to the system, as new objects might appear and drastically change the surroundings.

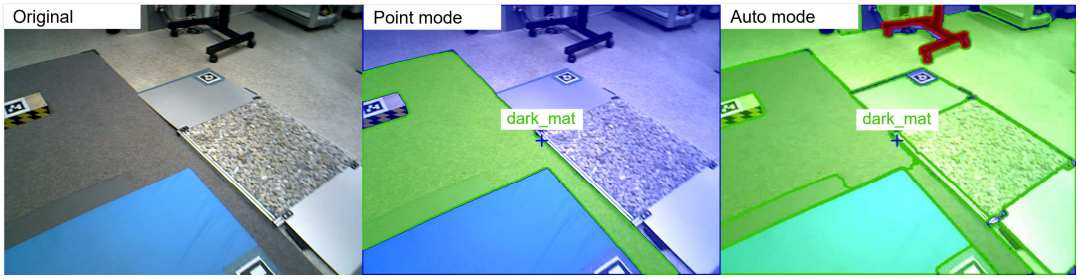


Figure 11: Example of both SAM-Track modes: point (middle panel) modes and auto (right panel). In green, segmented and classified objects; in red, segmented and unknown objects; in blue, not segmented objects.

The automatic mode also requires far more memory than the interactive one. The method has to save information of the segmentation for every object in the scene, but also for the tracking, which spikes the memory consumption. Depending on the setting this issue might not be a problem, but if we focus on mobile robotics, most of the times it is wanted a efficient method that does not exhaust the few available resources.

On the other hand, the automatic mode provides information of the whole environment. That is, by segmenting all the objects present in the frame, the robot can process all the information of its surroundings and be fully aware of how to proceed. This is not the case for the interactive mode, as the method only has the mask of a single object. In the case of a walking or grasping robot, however, it might be enough to just focus on the terrain or object ahead of them, as their functionalities are also limited.

3.1.2 Model checkpoints

As SAM-Track is formed by SAM and DeAOT, there are different models for each tool. In the case of SAM, there are three model versions available with different backbone

sizes: huge, large and big. The accuracy of the method heavily relies on which model is used, but we have to take into account the availability of memory. Loading the huge model takes more than 2 GB of GPU, while the smallest one only takes up to a few hundred MB.

Regarding DeAOT, we find much more variety. Some of them have their pre-training stage with static images, while others have used for the main-training stage DAVIS [45] and Youtube-VOS [60], which are known large-scale VOS benchmarks. All of the available models have been compared and evaluated in the datasets, but they have also measured the frames per second that each one is able to achieve. This is very relevant in real-time robotic scenarios, in which we have a trade-off between the accuracy, fps, and model size. In that way, heavier models might achieve a better tracking accuracy, but they are not able to run many fps. On the other hand, a light model will process many fps, but will have more errors in the tracking. For DeAOT, we can find six different models with different parameters size, ranging from tiny to large, and there are twelve in total depending on the training phase. Their accuracies go from 82% for the lightest model to 86.2% for the heaviest one, and they are able to achieve a maximum of 53.4 fps and 11.9 fps, respectively, evaluated in 4/1 NVIDIA 100 GPUs.

Depending on the setting, different model combinations can be used. We could focus on the segmentation or the tracking by using a heavier model in each task, but in a real-time mobile robotic system we prioritize the memory and energy consumption, and the processing time, without degrading in excess the accuracy of the method. Besides, a high fps rate is desirable, but for instance the human eye is believed to see from 30 to 60 fps [1], which is the rate at which most screens are refreshed so that we see a continuous stream of light instead of flickering images. However, on a possible robotic application the objects might be static in the environment. Even if we take into account the own movement of the robot, it might not need such high frame rate to be aware of what is happening in its surroundings. It depends on the application, because others such as autonomously driven cars will need a very high fps processing for safety reasons.

For the experimentation part of this thesis, we will use a NVIDIA RTX A2000 12 GBytes Graphic Processing Unit (GPU). This memory allows to process the bigger models of SAM and DeAOT, so we will study the best performance rather than the lightest one. With this, we will achieve to compute the maximum GPU usage and the minimum fps of the performance. For mobile robotics, it is necessary to save as many resources as possible, so the point of using the biggest models is that we can always use smaller ones to alleviate the GPU. The actual performance of the method, however, does not have to be downgraded with the model, but the results might differ between them, as they have a different number of parameters. Specifically, we are using: SAM model type "vit_h" weighing 2.4 GBytes, and DeAOT model type "R50-DeAOTL" weighing 225 MBytes.

3.1.3 Parameters and inputs

SAM-Track aspires to be a foundation model for a wide range of CV applications, so that the method does not have to be trained for new practices. Nevertheless, apart from the model selection, there are some parameters that help to fine-tune the method and improve its efficiency according to the specific implementation.

In SAM, there are some parameters that can be modified in the case of automatic segmentation. They allow to change the segmentation results in a frame, although most of the times the ones by default work properly. The most relevant for our work are the following ones:

- `point_per_side`: number of points to be sampled along one side of the image.
- `point_per_batch`: number of points to run simultaneously.
- `thresh`: different thresholds for the predicted masks to modify the segmentation results.
- `crops_n_layers`: number of crops of the image to run the mask prediction.
- `min_mask_region_area`: post-processing will be applied to remove disconnected regions and holes in the masks according to this parameter.

For the interactive segmentation, there are not parameters that change the segmentation of the image. It just accepts different kind of inputs:

- `point_coords`: array of point prompts, each point indicated as position (X,Y) in pixels.
- `point_labels`: array of labels for each point prompt, indicating whether the point refers to a foreground object or to the background.
- `multimask_output`: option to activate if the model returns only one mask with the highest quality score or three masks with different score.

In the case of DeAOT, we can choose a few parameters regarding the frequency of memory update, although they are recommended to be high.

Lastly, SAM-Track has some particular parameters that also affect the performance of the method:

- `sam_gap`: interval to run SAM to segment new objects (only for automatic mode).
- `min_area`: minimal mask area to add a new mask as a new object.
- `max_obj_num`: maximum number of objects to track in a video sequence.
- `min_new_obj_iou`: minimum ratio of new object mask over background to consider a segmentation as a new object.

All of the mentioned parameters or inputs have to be carefully chosen so that the resulting segmentation and tracking are adequate for the specific application of the method. For most of the practices, the default parameters should return acceptable results, but sometimes they might need to be fine-tuned. For example, if certain frame is crowded with objects that are not big, the `min_area` parameter of SAM-Track should not be too high, otherwise some true object masks will be removed. However, in a planetary exploration context, the rover might focus on the terrain that it is walking on instead of on the individual rocks that it might encounter. Thus, the minimal area parameter should be set high, so that small objects are ignored but the terrain is detected correctly. Then, when grasping a rock, the parameter value could be decreased so that small objects are considered again in the segmentation and tracking process.

Parameter	**Value**
`point_per_side`	32
`point_per_batch`	64
`thresh`	0.8 - 0.9
`crops_n_layers`	1
`min_mask_region_area*`	8000
`point_coords*`	(320, 240)
`point_labels`	1
`multimask_output`	True
`sam_gap*`	N/A
`min_area*`	8 000
`max_obj_num`	255
`min_new_obj_iou`	0.8

Table 2: Parameters and inputs set for the different parts of the method: SAM, interactive mode, and SAM-Track. With (*), the parameters changed from default.

In this thesis, we will use for the most part the parameters by default, which are stated in Table 2. The minimum area, the minimum mask region area, and the point coordinates are the only parameters that we modified. The dataset that we collected from TORO is described in the following Section 3.4, and it contains the main terrains that TORO walks on. After several trials, we decided to modify the `min_area` parameter, so the segmentation returns preferably objects bigger than this value. If none of the results is bigger than the minimum set area, the mask with the higher area will be returned. Moreover, we will only focus on the point in the centre of the frame, as we stated in Section 3.1.1.

3.2 Continual Learning: Progressive Neural Networks

Regarding the CL part, there are multiple approaches as mentioned in Section 2.2 but PNN has been proven to be efficient in different projects at DLR. For that reason, in this thesis a PNN implementation will be used, described in [49]. This work was

also carried out in the company, and they used PNN combined with Bayesian Neural Networks in order to improve generalization bounds and uncertainty estimates. As a result, they obtained a variation of the algorithm, the Progressive Bayesian Neural Networks, a extension to the CL setting. For our work, we will be using the simple PNN algorithm, which increases the size of the network with each task by appending a copy of the base network to the current architecture. Additionally, lateral connections can be formed in order to transfer knowledge between previous columns and newly untrained columns.

Thus, the PNN consists of mainly two elements, the backbone and the base network. The backbone is used as a feature extractor so that the input, an image in this case, undergoes a dimensional reduction. This helps the base network in the sense that it will only focus on the most relevant features from the image and will be trained in a more efficient manner. On the other hand, the base network is the main NN, it will be replicated and trained for each particular task, so that the whole network is composed by multiple base networks. This base network can be a NN with any configuration of layers. PNN also implements a transfer learning and it can connect layers from previously trained networks to layers from new networks. However, this is not the focus of this thesis, as we are more interested in the permanency of knowledge rather than in its transfer from column to column, but it is an interesting feature to take into account.

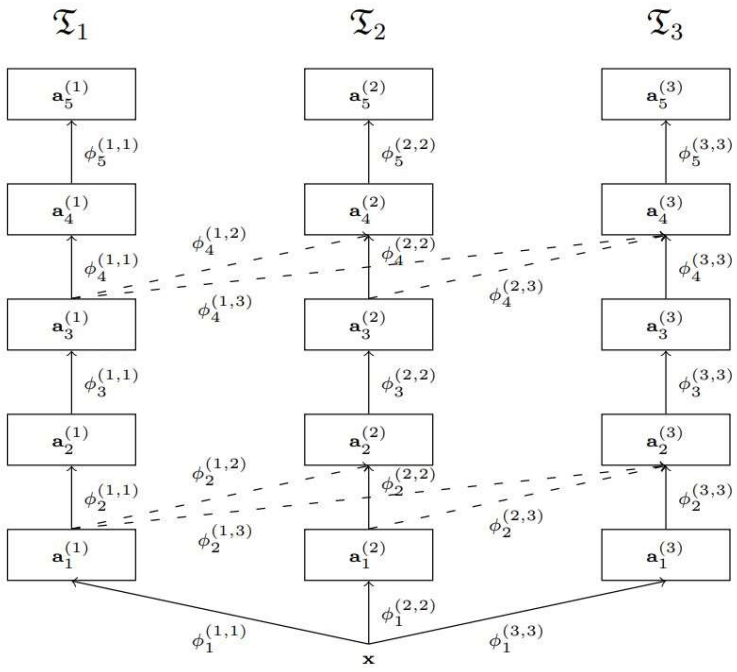


Figure 12: PNN diagram showing a 3-column and 5-layer network. The input is $\mathbf{x}$, each column process it and returns its output. Each column correspond to a different task, T_1, T_2, T_3, and there might be enabled lateral connections (dashed lines) [49].

In the Figure 12, a PNN with three columns and five layers per column is displayed. All PNN start with the basic NN for the first task, and every time a new column is

added for a particular task, the rest of previous columns and lateral connections are frozen so that only the current column is optimized using its output. Consequentially, the network does not forget previous tasks as their previous columns are not modified, and it can even use that former knowledge to achieve a positive information transfer.

3.2.1 Backbone: DINOv2

As backbone for the PNN we are using a recently released method called DINOv2 by Meta AI [41]. It is the evolution of its predecessor, DINO, aimed for training CV models. The first version was a system for unsupervised pre-training visual transformers, and it could visualize attention maps for random images or videos for image retrieval, segmentation or even zero-shot classification using a classifier in the extracted feature space. This new version is even more powerful and generates results with higher quality, as it has been trained with a larger and curated dataset and has improving modifications in the algorithm and implementation. In that sense, the model serves as a multipurpose backbone for many types of applications without fine-tuning. The output of DINOv2 can be directly used as input for any other purpose, for instance, image semantic classification. This method complements SAM in the CV research by Meta AI, and while the purpose of each other is different, they create horizontal impact in the field.

We are using this method in order to optimize the training and inference process, so that the input to the classifier is not a whole image with a huge amount of pixels, but a reduced number of relevant features extracted from it. Currently there are multiple algorithms and methods that carry out the same task as DINOv2, but we decided to use it in order to follow a line of work in which we approach newly state-of-the-art models that constitute the foundation of many wide ranging applications in CV, in the same way as SAM. There are different vision transformer models available, the main one with a billion parameters, while the rest are distilled from it. The input image for the model has to be 224x224 pixels, or a multiple of the patch size, which is 14. The output of DINOv2 returns a class token and patch tokens, and the dimension depends on the model, ranging form 384 for the smallest to 1536 for the biggest one. In our method, we just pass directly these extracted features to the next step of the PNN, which is training a new network for a given task.

3.2.2 Base network: linear classifier

PNN is quite flexible when referring to the base network, since it can be any NN. In DINOv2 documentation [41], it is stated that the output of the encoder can just be used with downstream classifiers such as linear layers, and the results would be competitive. For that purpose, in this work we have implemented a very simple linear classifier NN with a few hidden layers. The input of the classifier is the output of the backbone, so it will depend on the DINOv2 model that is used.

This part has been defined by trial-error, we have been testing different configurations of hidden layers to see which one had a better accuracy and efficiency in classifying images processed by the backbone. As a result, we obtained a neural

network with 7 hidden layers, ReLu activations (the output is the input if positive, or zero if negative), and a final linear layer with one single output. A diagram of the resulting NN is shown in Figure 13, with the different layers that form it.

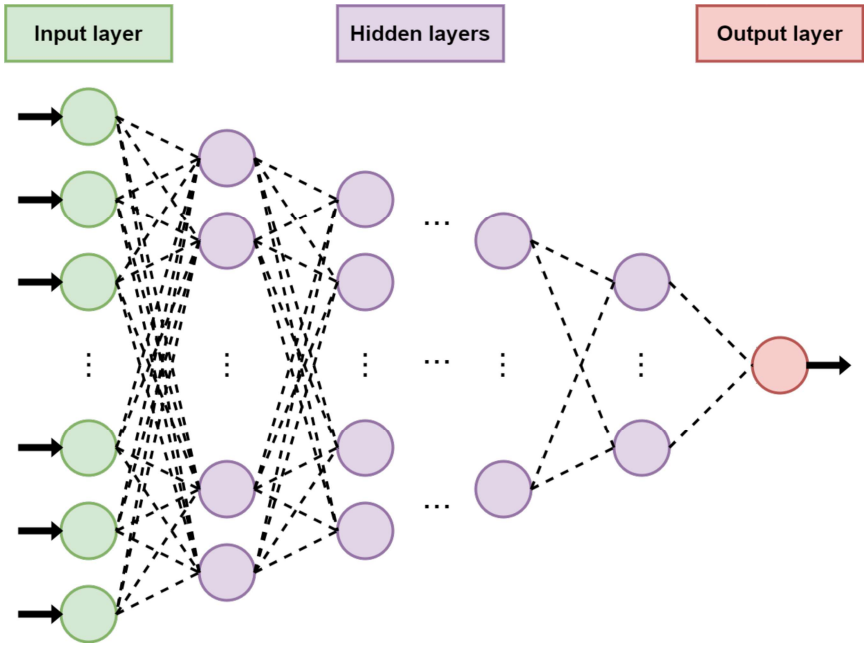


Figure 13: NN diagram used as base network for classification in PNN.

To train each one of the NN, there are some parameters that can be modified and influence the resulting network. These parameters are commonly known in the ML field, as they regulate a lot of commonly used DL implementations [49]:

- `learning_rate`: controls how slow or fast the network adapts to the problem, it is a value between 0 and 1. A large value makes the model converge too quickly to a sub-optimal solution, while a very small value can get it stuck.
- `weight_decay`: used to prevent overfitting, is a regularization technique that adds a small penalty to the loss function.
- `num_epochs`: the number of times that the training dataset passes through the algorithm.
- `patience`: value that dictates the number of epochs after the loss has stopped from decreasing to stop training. It helps to reduce training time when there are no improvements.

For the sake of simplicity, we will fix these values taking into account previous works [49] and fine-tune them. After this process, we get a configuration that works acceptably for almost any experiment: learning rate of 0.001 (that can be changed mid-training with an optimizer, if the loss is low enough), weight decay of $1 \cdot 10^{-8}$, patience of 20 epochs, and 50 epochs per training.

Both this NN of the PNN and the backbone can be changed for any other combination, depending on the application and the nature of the problem. For that reason, and in order to validate the choice that has been made in this thesis, we will test different configurations in Section 4.1.3 and obtain results on the performance of the PNN, so that our initial idea is supported with results. However, the intuition behind this is that DINOv2 is a very versatile and powerful feature extractor, so that a simple classification network would suffice for the purpose of this thesis.

3.2.3 From task-incremental to class-incremental PNN

One of the main goals with this method is not only to classify known objects with the base network, but also introduce new unknown objects in the system so that the robot can learn continuously. With a linear layer we can do the former, and the PNN enables the latter, but the part that detects unknown objects is missing. This is a common problem in ML, and it is defined as OOD detection [61], briefly mentioned in Section 2.3.1. As it is not the main focus of this thesis, we will implement a very simple but working approach, keeping in mind that this is an area of the method that might be improved in the future.

To begin with, it is remarkable that PNN is a task-incremental algorithm [49]. That is, it is assumed that all the tasks are arriving sequentially and the previous ones cannot be accessed by the method. This is also true in our setting, as the method aims to learn new objects in a sequence, and previously learned objects will remain inaccessible after the network has trained its respective column. In these type of algorithms, the task ID must be known both during training but also during testing. Here we encounter the first issue; in a real-world setting with mixed both known and unknown objects, it is difficult to know beforehand to which task ID the new task belong, if not directly stated by a human supervisor. This case corresponds better to what is called a class-incremental learning, which is when the task ID is only known during the training, but not for the inference. In this sense, each column is trained sequentially and we know the class trained in each column, i.e. the task ID, but when doing inference, we do not know the class of the object and we are ignorant of its task ID. In order to solve this, we implemented two changes to the original PNN algorithm.

In the first place, we have to train each column with objects belonging to the same class, as we do not have information of the objects that might belong to an unknown class. We aim to train each NN as a binary classifier so that they know whether the given inferred object belongs to that class or not. For this, we can either use a Cross Entropy loss function – training only one out of two outputs, the known one – or directly a Binary Cross Entropy loss function that achieves the same objective and is a particular case of the former. We will use the Pytorch [43, 53] library for Python, which already implements both functions. They take the outputs of the NN, the unnormalised logits of the classifier, and compute the unreduced loss described as:

$$\ell(x, y) = L = \{l_1, \cdots, l_N\}^T, \tag{1}$$

where the inputs are designated by x, while the target labels are y. The N describes

the dimension of the input batch. Then, the total loss is computed as the average value of the array $\ell(x, y)$.

To process each one of the loss elements, the main formula used for the Cross Entropy loss function is

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})} \cdot y_n, \tag{2}$$

which is useful when training a classification problem with C classes. Here, the weights w can be used to assign more or less importance to a given class. A particular case is found for binary classification, when we train the NN to classify either in one or another class, so the same formula can be reduced to

$$l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]. \tag{3}$$

In this case, the inputs x_n are expected to be probabilities between 0 and 1, instead of the logits outputs of the NN. For this reason, there is another variation of this loss function, called Binary Cross Entropy with logit loss, in which both a sigmoid layer and the loss function are combined so that $\sigma(x_n)$ is processed instead of x_n, taking advantage of the log-sum-exp trick for numerical stability.

Once a classification head is trained using a loss function, it will return the unnormalised logit values for every input image. These values are then commonly used in different OOD detection methods in order to identify which samples are in-distribution and which are out-of-distribution. Therefore, in this thesis we will compare the performance of two OOD methods that are efficient yet simple to implement. The first of them is the Maximum Softmax Probability (MSP) baseline [26], that makes use of the negative maximum softmax probability as the anomaly score in the following way:

$$a = -\max_k \frac{\exp f(x)_k}{\sum_i \exp f(x)_i} = -\max_k \hat{p}(y = k \mid x), \tag{4}$$

where $f(x)$ is the output of the network, that is, the unnormalised logits of classifier f on input x. In a system with more than one output, such as one trained with a Cross Entropy loss function, this value would be computed as the maximum softmax probability out of the k returned logit values from the classifier. This method is based on the principle that classifiers tend to have more confidence on samples that are in-distribution, rather than OOD ones, so we would get higher probabilities.

The other method that will be evaluated is called MaxLogit [25], and directly uses the unnormalised logit values as an anomaly score. Concretely, the maximum returned value among the different logits:

$$a = -\max_k f(x)_k \tag{5}$$

The idea behind this is that the softmax probability can be affected if two classes are visually similar, so it might return a lower confidence not because the objects are OOD samples, but because the exact class of them is difficult to ascertain. By using

the logits, we assure that the anomaly scores are not affected by the number of classes, and the detection OOD might be improved.

Therefore, we will train a column with one single category using the train dataset and either a Cross Entropy or a Binary Cross Entropy loss function; then, with the validation dataset, we will obtain the reference in-distribution anomaly scores which will be used to detect OOD samples in each column. When doing inference, we will obtain logit results from each one of the trained columns, and we will compute the anomaly score of the testing samples with them. Depending on the results, we will determine if the samples are in-distribution of any of the trained columns (if the values are similar to the validation anomaly scores), or if we have encountered an OOD sample that belongs to an unknown category. Using an OOD detection method and training binary classifiers for each category, we achieve transforming the initial task-incremental PNN into our objective, a class-incremental algorithm that does not rely on the task ID of the samples in order to obtain a classification result.

3.3 Integration of SAM-Track and PNN: SaLS

Both VOS and CL methods have been integrated in the same pipeline, and we obtain a single architecture that joins the capacities of both powerful tools. The proposed method is Segment and Learn Semantics (SaLS), a perceptive system that updates online, aimed for locomotive systems such as walking robots or land rovers. The full architecture is shown in Figure 14, and it is illustrated with an example of its working.

The input to SaLS is a sequence of frames, in this example corresponding to $t = 0$ and $t = 1$ which describe a certain scene in two different moments. Starting with the first frame and depending on the selected mode, it can either be processed for Automatic Tracking or for Interactive Tracking mode. In the first case, SAM is run to segment everything in the frame, returning as a result the masked frame with the different objects for $t = 0$. These masks are then compared with the previous batch of masks in order to find new untracked objects. In the case of the Interactive segmentation, a point prompt previously set is used to generate the mask of one single object, so the output of SAM in this case is the mask according to the given point at $t = 0$. The resulting masked frame in both cases is then processed by DeAOT and the frame is added as reference in order to keep track of the segmented objects.

The previous phase is done when the segmentation of the frame is required, in the case of the Automatic mode that happens every `sam_gap` and in the Interactive mode when the point prompt is in an unsegmented area. If the segmentation is not performed, then the algorithm just track the objects in the new frame $t = 1$ from the segmentation done in $t = 0$. The result in both cases is a masked frame containing the different objects, either from segmentation or from tracking. These masks are then separated into the different objects, more than one in the Automatic case, or just one in the Interactive mode, denoted as $Mask$ 1, $Mask$ 2, etc.

At this point, the segmentation of the frames is complete, so now it is required to classify the detected objects using the PNN. Firstly, the object masks are processed by the backbone, which is DINOv2. The extracted features are then given to the multiple N NNs (NN_1, ..., NN_N) that constitute the PNN. Each one of them is trained on a

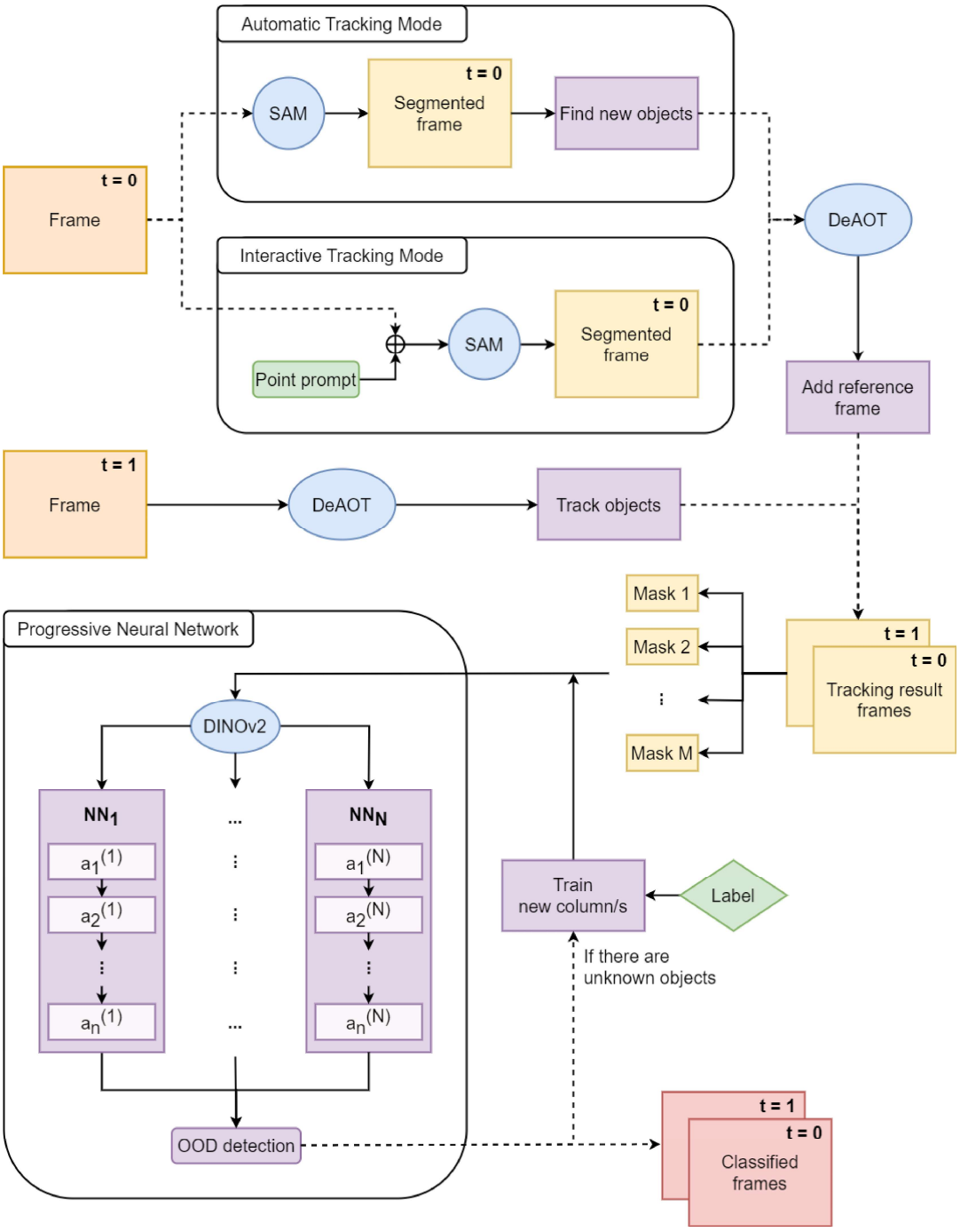


Figure 14: SaLS overview: includes SAM and DeAOT as part of SAM-Track method for VOS, and PNN with DINOv2 and NN for CL and image classification. In orange, the inputs to the system; in blue, the different external modules; in yellow, the intermediate results; in purple, the intermediate functions; in green, the user inputs; and in red, the resulting frames.

different object class but they all have the same number n of layers: $a_1^{(N)}$, $a_2^{(N)}$, ..., $a_n^{(N)}$. The outputs of each head are used to compute the anomaly scores and decide if

each object is assigned to an specific known class, or to the category of the unknown. In the latter case, the PNN is re-trained to add new columns for the unrecognised classes, given a label as an input by a human supervisor, and the classification of the masks is performed again, so that the new class or classes are included. At the end, the resulting frames $t = 0$ and $t = 1$ contain both the segmentation and the classification of the objects that appear on them, so that SaLS is able to add semantics to a given scene and learn continuously from the unknown.

The method has been coded using Python, given that the integrated tools are also coded in the same language, and employing PyTorch [43, 53] as the ML framework. A overview of the pseudo-code is displayed in Annex C. It is worth mentioning that the process is fully automatic, and the only user task is when the PNN is re-trained. This part is the only one in our method that actively demands a human interaction once the algorithm is run, apart from the previous modifications and adjustments such as the initial parameters, and the settings of the point prompt and the NN thresholds.

3.4 TORO dataset

For the experimentation part, we decided to use a new dataset acquired from the TORO robot directly. In this way, we can focus the development of the method on this specific application, and using images collected from it might help the fine-tuning. This dataset was conceived to have as categories the different types of objects that the robot encounters in its closed environment in the lab. In order to perform tests on its walking and other researches, there is a small area in the DLR laboratory with a circuit that TORO is able to walk as a demonstration of the robot capabilities in different types of terrains. In the Figure 15 we can see the robot standing and the different obstacles that it can sort out. There are 7 main objects that we can identify, on which the robot walks: laboratory floor, dark mat, blue mattress, grey mat, pedestal, ramp, and stones. In the dataset, we have combined the dark mat and the grey mat due to their very similar characteristics, it would be very difficult to distinguish them with the classifier. Moreover, there are no relevant differences on the way that TORO walks between both terrains, so for simplicity, they have been joined in the same category: dark mat.

Apart from these mentioned terrains, the robot also sees other objects in its demonstration walk, however, none of them are walkable. We include them in the dataset nonetheless, as they help the robot to understand its surrounding and might be useful to improve its perception of the walking terrains. These other objects are: bistro table, cables, extinguisher, PC table, person, airplane wall, stand board, TV mount, and wall. Currently, only the bistro table and the wall are also provided with AprilTags, but the rest of them are not.

Knowing this, we collected different image sequences in order to obtain a varied dataset of the same objects from different points of views. In order to speed up the process and due to the limitations on the usage of the robot, some sequences were recorded with the robot hanging, while others could be collected while the robot was walking. The resulting images in both cases are very similar, being the only difference the pace of walk and the disturbances that might appear when the robot walks a step,

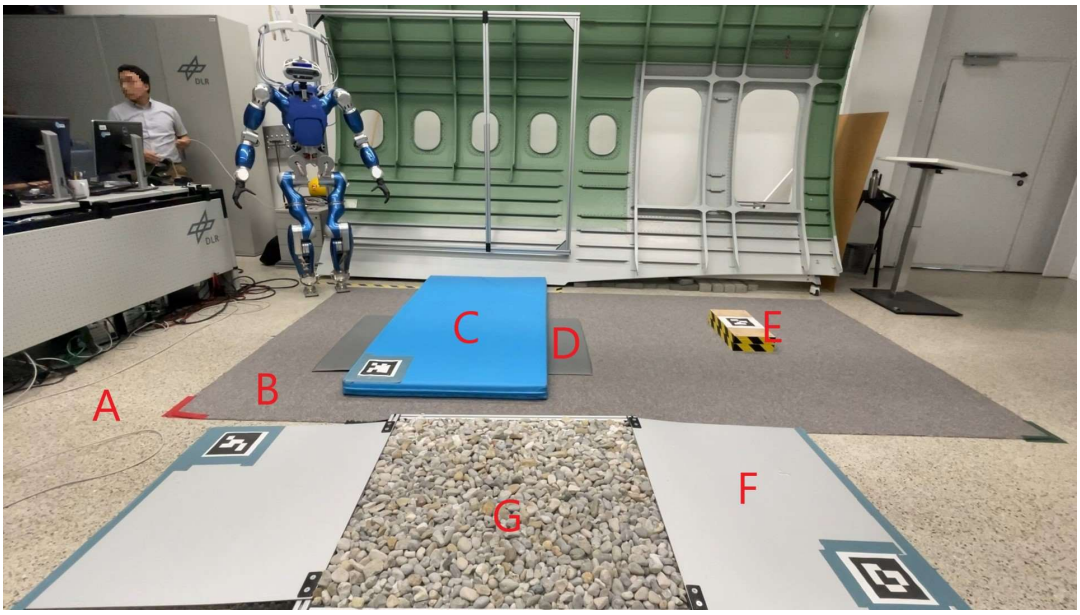


Figure 15: TORO walking circuit: (A) lab floor, (B) dark mat, (C) blue mattress, (D) grey mat, (E) pedestal, (F) ramp, (G) stones.

in the same way as humans do. Additionally, in the first case we could move the robot a bit faster than in the latter. Otherwise, all the recorded sequences could be interchangeable and are meaningful for our purpose.

In total, we obtained 7 different sequences at around 10 fps with a resolution of 640 x 480 pixels, 5 of them with the robot hanging and 2 of them with the robot actually walking. From the two walking sequences, one of them was captured as TORO was doing its usual automatic walking demonstration, and the other with TORO looking down (seeing the terrain and part of its feet), while controlling its movements with the remote controller. The sequences have from 200 to around 3500 frames, summing in total around 6000 frames (around 10 minutes of video). Using the method proposed in the thesis, SaLS, we have obtained the different object crops from these sequences and we have semi-automatically annotated them. Thus, we have identified the centre point-prompt segmentation object of every frame, assigned a label to it, and saved the cropped image. With this, we obtain the different 15 classes, with the 6 main terrains and 9 varied objects mentioned before. Some frames had to be corrected after the annotation, as the segmentation using SAM-Track is not perfect, but generally it yielded notable segmentation results that could be directly used for other purposes. This is another application of the method that was not considered beforehand, the semi-automatic annotation of video frames, and can be very useful for the creation of new video segmentation datasets.

We will use the objects obtained from the different sequences for the experiments, splitting them in different sets to reduce the bias, that is, for training, validation and testing of the method. We will combine for a same category different sequences for training and validation, and the rest of them for testing. The main purpose of the experiments is to validate the classification power of the CL method, so the

Category	Terrain	Sequences	Number of frames
dark_mat	✓	1 - 7	2 923
lab_floor	✓	1 - 7	1 110
stones	✓	1 - 7	218
ramp	✓	2, 4, 5, 6, 7	266
blue_mattress	✓	2, 3, 6, 7	346
pedestal	✓	3, 6, 7	218
wall	✗	1, 2, 3, 4, 7	61
PC_table	✗	2, 4, 7	63
plane_wall	✗	2, 3, 4	51
extinguisher	✗	1, 2, 4, 7	47
cable	✗	2, 4, 7	21
TV_mount	✗	1, 2, 3, 7	15
bistro_table	✗	2, 7	41
person	✗	5, 6, 7	20
stand_board	✗	2, 4	12

Table 3: Summary of categories of the dataset collected from TORO camera. It is indicated if the class is a terrain or not, the ID of the sequences that it appears on, and the number of frames that the object is shown.

segmentation will not be reviewed in the following sections, at least in a quantitative way. In the Table 3 we expose all of the categories that are present in the dataset, and in Annex A we display examples of each class.

It is noticeable that the dataset is unbalanced: from the 5 412 images that we have, two classes make up almost the 75% of the whole dataset. This is a common problem in ML, as it is difficult to always have the same amount of samples from every category, but for the purpose of this thesis this fact should not be relevant. In a real-world setting, we will train the network initially with the known categories, the ones that we know that the robot is going to encounter. For these classes, we might have a large amount of images available that we can use for training and validation. On the other hand, when the robot finds a new unknown object that was not initially contemplated in the training dataset, it will be able to learn it but just using the current single available image, so that there will be an unbalance in the classes. Nevertheless, the proposed method should work correctly in both cases, so we are going to operate using all the categories in the collected dataset.

3.5 Evaluation procedure and metrics

Once we have established the different parts that form the method and how they work, and we have collected and pre-processed the dataset that will be used during the experimentation part, we also have to set the procedure in order to train, validate and test the method, and which metrics we are using to evaluate its performance. We will focus in the classification power of the method, so we are not assessing the accuracy

of the segmentation part, which is extensively done in [11] and [31].

3.5.1 Evaluation procedure

We will perform two kind of experiments that require different evaluation procedures. The first set of experiments will focus on the decisions that we have made over the method in the classification side, i.e. the OOD detection algorithm or the backbone performance. In this case, we take the 6 main terrain categories described in Table 3 and distribute them in 6 different tasks that arrive sequentially. Therefore, the method will train each head also in a sequential way, in the same order that the tasks arrive. Then, all of the tasks will be evaluated after each training. In this way, we can obtain results on the CL ability of the PNN, and how the accuracy of the algorithm evolves while learning new tasks.

In order to reduce the bias, we evaluate the method over multiple iterations and initializing with different random seeds. Regarding the former, we achieve to use multiple random combinations of train-validation-test datasets so that the results do not depend heavily on the sequences from the dataset that we choose for each phase. To do so, every iteration we divide the dataset in the three phases – training, validation, and testing – choosing randomly `max_seq_data` images from every sequence that the object appears on. Besides, as every object appears in a different number of sequences, we tried to dedicate around 50%-60% (if the object appears in an odd or even number of sequences) of them for training and validation, and the rest of them for testing. Then, from the train-validation split, a 30% is used for validation and the remaining 70% for training. A large percentage is used for the validation split because it serves to obtain the anomaly scores that will be useful for the OOD detection, so we consider that is very relevant to have as many sample values as possible in order to reduce the bias on the validation set and be able to generalise the anomaly scores to the testing set.

The second set of experiments is focused on the different sequences captured from the TORO camera, so here we will run the full method (segmentation and classification) over all the frames of each sequence. Consequentially, for each frame we will obtain the segmented mask of the object in the centre point prompt, and its classification using the PNN. As we stated in the Section 3.4, these sequences have been previously manually annotated with the ground truth, so we know the label of the object in the centre frame. As a result, we are able to compare the performance of the whole method in the classification aspect with the true labels of each frame, for every available sequence.

In this case we will also evaluate the method over multiple iterations and initializing with different random seeds. However, when doing the split for random combinations of train-validation sets, we will not take into account the sequence that is being evaluated, so that the objects appearing on it are totally new to the method. In this way, we ensure the proper working of SaLS, in a very similar way to how it would be used in a real-setting: we would train the PNN with collected images from the known objects, and then use the method in a new environment which may or may not contain the same objects, but surely will contain objects never seen before. The percentages for

training and validation are kept the same as we described in the previous paragraphs, so that we have multiple objects to compute the anomaly scores. Evaluating this part of the experimentation, we will obtain results of the classification performance on both the known objects, but also on the unknown ones, and if the training has been successful or not, since we are doing the validation of the method frame by frame.

3.5.2 Metrics

In order to to compare the performance of the proposed method with other state-of-the-art works, it is necessary to use standardised metrics that are able to account for the CL setting and show numerically how good or bad the continual classification is. Given the novelty of the topic at hand, it is difficult to find a general consensus, but there are some recent researches that study in more detail this issue and propose new frameworks to evaluate CL algorithms [16, 36, 51]. Therefore, we present the main metrics that will be used to evaluate the experimentation part.

$a_{i,j}$	T_{e_1}	T_{e_2}	$\cdots$	$T_{e_{N-1}}$	T_{e_N}
T_{r_1}	$a_{1,1}$	$a_{1,2}$	$\cdots$	$a_{1,N-1}$	$a_{1,N}$
T_{r_2}	$a_{2,1}$	$a_{2,2}$	$\cdots$	$a_{1,N-1}$	$a_{2,N}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$T_{r_{N-1}}$	$a_{N-1,1}$	$a_{N-1,2}$	$\cdots$	$a_{N-1,N-1}$	$a_{N-1,N}$
T_{r_N}	$a_{N,1}$	$a_{N,2}$	$\cdots$	$a_{N,N-1}$	$a_{N,N}$

Table 4: Accuracy matrix result of evaluating the test set of task j after training with the training set of task 1 to task i, assuming N tasks in total [16]. In blue, the accuracy of tasks previously learned with respect to the current trained task; in yellow the accuracy of tasks not yet learned with respect to the current trained task; and in white in the diagonal, the accuracy of the task corresponding to the current trained task.

Accuracy ($a_{i,j}$). This metric is a ratio between the number of correct predictions by the classifier and the total number of predicted samples. In a CL setting where we train different tasks sequentially, we can compute the train-test accuracy matrix shown in Table 4. Here, we define $a_{i,j}$ as the accuracy evaluated on the test set of task j after training the network from the first task 1 until the task i, being N the total number of tasks. As an example, $a_{1,2}$ refers to the accuracy of the model on the test set of task 2 after having trained it with task 1. Similarly, $a_{2,1}$ indicates the accuracy on the test set of task 1 after training the model with the training sets of tasks 1 and 2. This is the basic metric from which we can define some others, based on the accuracy matrix.

Average Accuracy (A, a_{e_i}) represents the average accuracy of the model after training with the task i, and evaluating on the test sets of tasks 1 to j. This metric denotes how well the model performs on the tasks that it has been trained on. If $i = N$, the accuracy represents the performance by the end of the training, testing with the whole sequence. This metric is not only averaged by the number of trained tasks, but it takes into account the accuracy at every timestep to better define the CL performance.

$$A = \frac{\sum_{i \geq j}^{N} a_{i,j}}{\frac{N(N+1)}{2}} \tag{6}$$

Apart from this averaged accuracy across all the training tasks, we can also compute the average accuracy for each test task i, taking only into account the results after training with the same training task i. In Table 4, for example, we would compute a_{e_1} as the average of the values from $a_{1,1}$ through $a_{N,1}$; and a_{e_N} with just $a_{N,N}$. This metric will help us to know the real average per test task of the model, including improvements or deterioration of the performance for a given task with the learning of new tasks:

$$a_{e_i} = \frac{\sum_{j \geq i}^{N} a_{j,i}}{N - i + 1} \tag{7}$$

Backward Transfer (BWT) indicates how learning a new task influence the performance on previous tasks. It is very related to the concept of "catastrophic forgetting", given that it measures the ability of the model to not degrade the performance on previous tasks, that is, to not forget them. It is computed as the accuracy on a test set T_{e_i} after learning T_{r_i} and at the end of the last task on the same test set. If we expand it to consider the BWT after each task, we would use all the elements in the blue cells in Table 4. The equation would then be:

$$BWT = \frac{\sum_{i=2}^{N} \sum_{j=1}^{N} (a_{i,j} - a_{j,j})}{\frac{N(N-1)}{2}}. \tag{8}$$

In this way, positive values of this metric are considered to be beneficial backward transfer, and negative values would mean that catastrophic forgetting has taken place. To map BWT so that it ranges between 0 and 1, this metric is divided in two different concepts: the Positive BWT (BWT^+)

$$BWT^+ = \sum_{i>j}^{N} \frac{a_{i,j}}{\frac{N(N-1)}{2}} = \max(BWT, 0), \tag{9}$$

and the Remembering (REM) – originally negative values of BWT,

$$REM = 1 - |\min(BWT, 0)|. \tag{10}$$

There is another metric related to the BWT, the Forward Transfer (FWT). In this case, it measures the influence that learning a new task has over the performance of tasks that have not yet been trained. That would correspond to the values in yellow in the Table 4. In this thesis, we do not expect models trained on previous tasks to get any level of accuracy at all, given that they are supposed to be considered OOD samples. For this reason, we are not using this metric in our work, although it would be interesting to explore how this could be implemented in the method in future research.

Lastly, we will use one last metric related to the accuracy, which will be essential in the last part of the experimentation.

Confusion matrix is a table that contains the number of observations known to belong to a certain category i and predicted to be in category j. It displays in a very simple way the classification performance of the algorithm, since we can directly see which elements where classified correctly (diagonal elements), and which categories might be confused between them. Some other metrics can be derived from the confusion matrix, but for simplicity, we just will analyse the table after the classification. One example is shown in Figure 16, with three different categories: "0", "1", "2". The left panel is normalized over the total number of objects per category (by row), and the right panel is the plain number of elements. The higher the number in the diagonal elements, the better the classification performance.

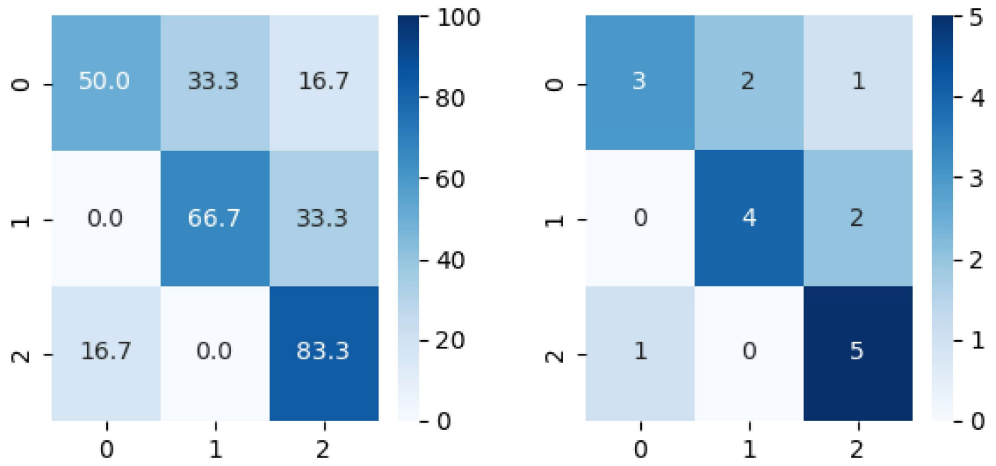


Figure 16: Confusion matrix examples, normalized by the number of elements per class (left panel) and non-normalized (right). The true labels are on the y-axis, while the predicted labels are on the x-axis.

We will also take into account other parameters not related to the accuracy on the test datasets. These ones are mainly focused on the consumption of computational resources, such as the size that the model takes up, or the time. These metrics are extremely relevant in a mobile robotics context, as we are aiming for the fastest and computationally cheapest, yet the most efficient model. Most of the times mobile robots do not possess huge hard drives to store thousands of networks in order to classify the objects in their environments, or very fast clocks to process a large amount of instructions, so these are crucial metrics to assess if the method can actually be used in the real world.

Model size and number of parameters. One of the most direct measurement of how much the model takes is its size in Bytes. With this, we can straight forward know if it fits in the hard drive of a robotic system. Most importantly, we are using PNN – a parameter isolation method that increases its size by adding a new NN for each task –, so that we know that our model will be taking more and more space for each task. This is closely related to the number of parameters, as every time that we add a new column to the PNN, we are increasing the number of parameters of the whole

network. It is worth mentioning that in the case of our method, both the model size and the number of parameters increase linearly and the same amount every time a new task is added. This is because both metrics are determined by the configuration of the NN that we are using a base network of the PNN. Nevertheless, it is still interesting to measure them and evaluate if they fall in an acceptable range or not.

GPU usage. Normally, NNs are trained using GPUs, as they accelerate the calculations due to the use of parallel processing, dividing the tasks in smaller subsets that are distributed among the cores. However, this is often a limited resource, especially in small robots which might not even have a GPU processor. For this reason, we also want to measure this metric too analyze how much GPU is needed in order to train the model under different circumstances, so that it can be a limiting characteristic in order to decide one model or the other.

Training time (t_{r_i}), that is, the time that takes to train the model with the training set of task i. This parameter will mainly depend on how many images are in the training dataset, and how fast the loss function converges.

Inference time (t_{e_i}), the time that it takes to evaluate the testing set of task j after training with the training set of task 1 through i. We can obtain a similar matrix to the one presented in Table 4 but for the inference times of each train-test pair, and in the same way as the training time, we expect this metric to change according to the number of samples in the inferred dataset. However, the inference time of a test set with the same number of images is constant, so the inference time will be more or less constant too. For that reason, we will compute the average inference time per task instead, so we will end up with a one single value of inference time for each task, with its respective standard deviation.

Lastly, there are a few metrics that are relevant in the field of OOD detection, and we will use them in order to compare different configurations [25]. These metrics are computed taking as input the anomaly scores calculated using the Equations 4 and 5, and distinguishing between the ones that belong to in-distribution samples (in this case corresponding to the negative samples) and the ones that belong to OOD samples (the positive samples).

False Positive Rate at 95% recall ($FPR95$) is the probability that an in-distribution sample is misclassified as an OOD sample when the positive rate is 95%. A model has a better performance when this metric is low, as that means that less percentage of in-distribution samples will be considered OOD. It is computed as the ratio between the False Positives (FP) and the sum of FP and True Negatives, having in mind the threshold at 95% imposed in the recall. In other words, $FPR95$ indicates the probability of false alarm when the probability of detection is 95%:

$$FPR = \frac{FP}{FP + TN} \tag{11}$$

Area Under the Receiver Operating Characteristic curve ($AUROC$) is the probability that an OOD example is given a higher anomaly score than an in-distribution sample. Therefore, we expect a higher value of this metric in the model, given that a random detector would have an $AUROC$ of 50%. It is computed using the scikit-learn

library [44]. Firstly, it calculates the receiver operating characteristic curve that pictures the performance of a classifier as its discrimination threshold is changed. Its axes are the True Positive Rate (TPR) versus the FPR, and finally, the area under the resulting curve is the $AUROC$.

Area Under the Precision-Recall curve($AUPR$) computes the average precision from the prediction scores. It sums up a precision-recall curve as the mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. In the same way as the $AUROC$, it is computed using the scikit-learn library [44], and a higher value is desired as it means that the model is more precise. Moreover, both metrics are very important because they give a holistic measure of the performance when the threshold for detecting OOD is not set yet.

OOD average accuracy (A_{OOD}). As an additional metric, we want to use a score that indicates how many of the samples are correctly labelled as OOD samples. In this way, after a prediction, we can compute the OOD accuracy as the number of objects correctly labelled as "Unknown" divided by the total number of true OOD samples. Consequently, we can obtain the same matrix as Table 4 but with the results from the OOD samples. We only expect to have positive results on the yellow area of the matrix, as it corresponds to the accuracy scores of the model after training with the task i and evaluating on the test sets of tasks i to N, that is, tasks that have not be learned yet. The average OOD accuracy is computed then in the following way:

$$A_{OOD} = \frac{\sum_{i<j}^{N} a_{i,j}}{\frac{N(N-1)}{2}}. \tag{12}$$

4 Continual Learning validation with TORO dataset

After the definition of the method – its different parts and how they work – and the establishment of the evaluation metrics, we can now proceed to run the experiments designed to validate the performance. Therefore, we can divide the experimentation part in two groups. The first one includes those experiments focused on the validation of the decisions taken in the CL part of the method, such as the crop size of the data transforms applied to the images before training, validation and testing; or the OOD detection method that works better for the given dataset collected from TORO. These experiments give insight into the characteristics of the method proposed in the thesis, and answer to the questions on why we have taken some of the most relevant design decisions in order to put together and efficient classification method that learns continually for mobile robotic systems.

The second group of experiments will be performed using the whole method, that is, including both the segmentation and the CL parts. In this case, we will test the performance of the proposed method using the collected and annotated sequences from the TORO camera. We will pre-train the network with the known objects – the walkable terrains – and run the sequence images. Hence, if a new object is encountered, it will be learned so that we can also test the CL capability in a real-world setting. We can then compare how the classification was in relation to the ground truth of the frames, and if the new objects were correctly assimilated. With these experiments, we want to validate the whole framework showing a possible real application and the ability of SaLS to operate in a mobile robot in real time, learning online while segmenting and classifying the objects in its surroundings.

4.1 Continual Learning methodical experiments

The first group of experiments focuses on the CL part of the algorithm. We perform different tests to validate some relevant decisions that affect the performance of the PNN: the size of the crop transformation applied to the images for training, testing and validation; the OOD detection method used to distinguish between known objects and unknown ones; and the backbone and base network configuration that works better for the classification with PNN. After these, we also evaluate the whole performance of the CL algorithm, given the decisions made on the previous described parts. Lastly, we execute a experiment related to the robustness of the method, in order to see how it works if we keep increasing the number of tasks.

4.1.1 Crop size in data transforms

In a first stance, before training the PNN we have a dataset with lots of images that come in a raw state. That is, they have different sizes and color distributions, and the objects in them might appear in different positions. Consequently, it would be very difficult to train and infer them, so a pre-processing step is commonly applied with the objective of normalising the input images. There are multiple variations of these transformations, depending on the nature of the dataset, but the usual ones are

cropping the image and normalising its colors. The latter is quite simple, as it just needs the mean and standard deviation of each color channel from the whole dataset. The former, however, could be thought as an hyper-parameter. We can choose virtually any size for the cropped image, but the performance of the algorithm greatly varies.

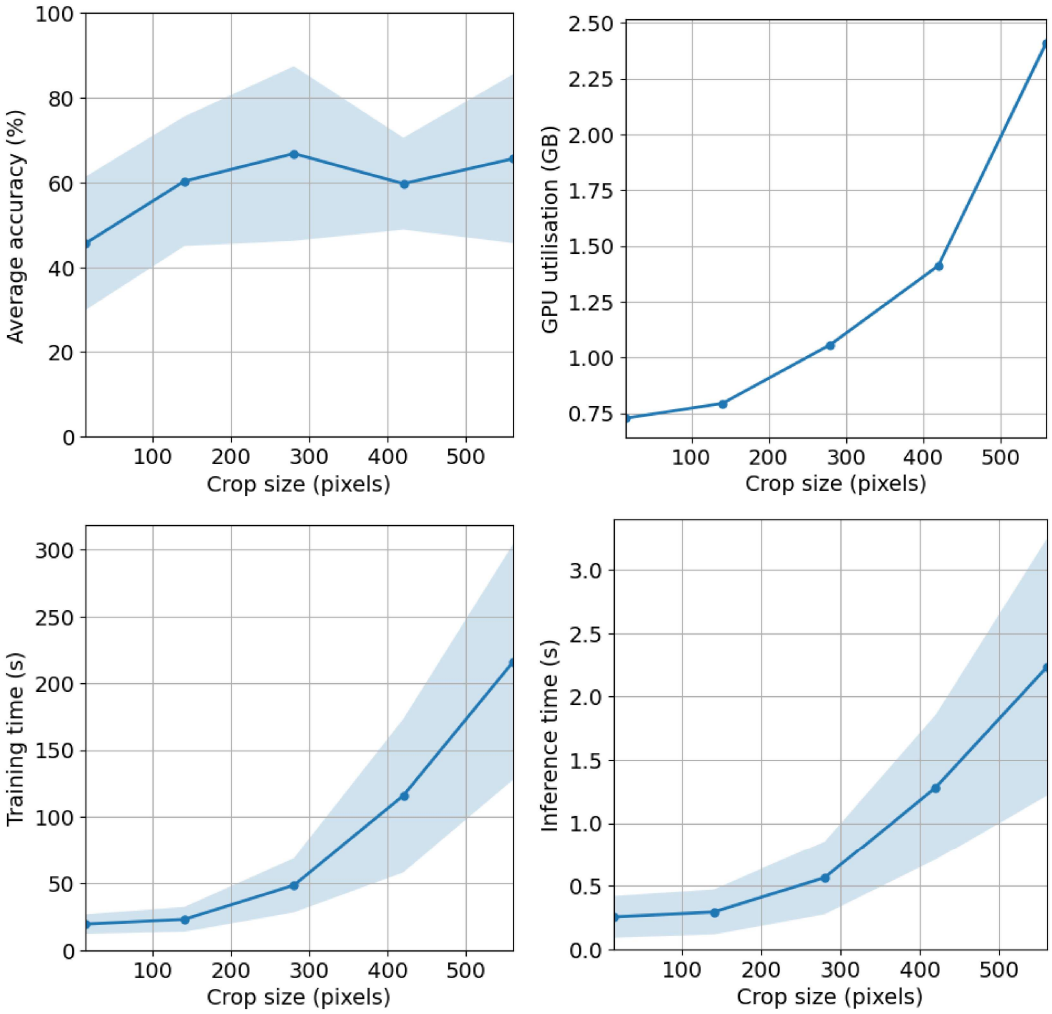


Figure 17: Results of average accuracy (top left), GPU usage (top right), and training (bottom left) and inference (bottom right) times in function of the crop size of the input images. The mean value across all the tasks and different seeds is plotted, accompanied by the standard deviation in a blue shade.

Consequently, the first experiment will verse on the determination of an acceptable crop size for the data transform that will be applied to every input image, either it belongs to the training, validation or testing set. We run only the PNN and use the main terrains of the TORO dataset as categories. After training sequentially each task, we evaluate each column with the test sets of all the tasks. As we are executing the code using the GPU, we also measure its consumption to check if it is affected by the hyper-parameter in hand. Other two important metrics for this section are the training and inference time. In this case, we take into account the average training

and inference time of all the tasks, as we are only interested in the variation of time in function of the crop size. To measure these metrics correctly and make sure that none of the other parts interfere in the results, we will be using the same working configuration: DINOv2 as backbone and a NN linear layer classifier. Even if the whole PNN is not fine-tuned, it is enough for setting this hyper-parameter and analysing the best performance.

In the Figure 17 we expose the mentioned metrics, and all of them are shown in function of the crop size, which varies from 14 to 560 square pixels, in steps of 14 square pixels. It is done in this way because the input of the DINOv2 only admits this input images with a resolution multiple of 14. However, it can actually be extrapolated to any value in between as we are using these values as a reference.

Starting with the average accuracy, on the top left panel Figure 17, it increases up to around 70% at a crop size of 280 pixels, and from that value on there is not a tangible improvement. For very small crop sizes, the images would not be very distinguishable one from other, as they would only have a resolution of 14×14 pixels. When reaching a size of 300 pixels, the cropped images are big enough to extract meaningful features from them so that the accuracy can increase, but it gets to one point where it does not get any better. The influence of the crop size can be justified to be meaningless from 300 pixels on, so that there is no point from the accuracy perspective to have larger image resolutions.

In regards of the GPU usage in the top right of Figure 17, we find that it increases exponentially with the crop size, going from around 750 MBytes when the images have the minimum resolution, up to more than 2 GBytes when the transformed images have 560×560 pixels square. This is a critic metric, because in mobile robots the available GPU memory might not be too large. It is worth mentioning that not all the consumed GPU memory is occupied by the transformed images, as the PNN network itself and the DINOv2 models are also moved there but always with the same size in all of the runs. Moreover, there is no standard deviation measure in this case, meaning that in all of the tested seeds the GPU usage was the same at every crop size. This makes sense given that this metric depend on the number of images in the dataset, which is the same for every run even though the images on it are randomized.

Lastly, both the average training and inference across all the tasks in the bottom of Figure 17 (left and right, respectively) have a very similar behaviour, tending to an exponential function. The time metrics depend on a lot of factors, but having in mind that most of them have been constant during the testings, we can clearly state that a larger input image to the network is translated to both a longer training process and an inference phase. Once again, it is also notable that in a real-world robot the processing times should not be very long, so that it is able to perceive, process and generate and answer to external stimuli. A very long inference time leaves the robot unaware of its surroundings, which depending on the application might be critical, e.g. an autonomous driving car. In the case of the training, this phase could be parallelised and run in the background, but it is still crucial to shorten it and learn efficiently as fast as possible.

Although the crop size hyper-parameter might not be so critical for the correct working of the whole PNN, it was necessary to evaluate its impact on the algorithm

performance. Therefore, we can state that there is no real improvement in having a crop size larger than 300 pixels, because the accuracy is not increased significantly and the computational resources consumed are boosted exponentially. To have a multiple of 14, in order to use DINOv2 as a possible backbone, we set this hyper-parameter in 280 pixels. At this value, the accuracy reaches its local peak, while the GPU utilisation and the training and inference times are kept at low and acceptable values.

4.1.2 Out-Of-Distribution detection methods

One of the most relevant points in the proposed method is the ability to separate the known from the unknown while training and classifying task-agnostic objects. As we described in the Section 3.2.3, turning the PNN algorithm from task-dependant to task-agnostic is an important challenge, and the way to tackle this is to train different heads for different classes and adding an OOD detection method. Regarding the latter, we described two different methods to compute the anomaly scores, the MSP and the MaxLogit, in Equations 4 and 5 respectively. Following the evaluation procedure presented in [25] and the metrics described in Section 3.5.2, we are going to execute an experiment to analyse which of the two methods fits best with our method and our dataset.

For that purpose, we will train the PNN algorithm with a basic setting similar to the previous experiment (DINOv2 and a NN as classifier), using both Cross Entropy and Binary Cross Entropy loss functions. Again, we will only use the main terrains from the dataset and will divide the dataset randomly to obtain train, validation and testing sets. The test sets will be used to compute the anomaly scores according to each method, and then we will calculate the metrics related to OOD samples to see which performance is better for our given dataset; those are the $FPR95$, the $AUROC$, and the $AUPR$. The tasks are trained sequentially, so that the current and the previous tasks are considered in-distribution samples and the rest of the tasks are OOD. We use both types of loss functions because even though they are similar, the Binary Cross Entropy loss is calculated directly with the logits instead of the output of a sigmoid layer. Therefore, we want to see if there is any improvement in the performance side, or it is only a computational advantage.

	CE		BCE (logit)	
	MSP	MaxLogit	MSP	MaxLogit
$\downarrow FPR95$	56.30 ± 10.48	26.27 ± 7.68	100.0 ± 0.0	**20.63 ± 4.21**
$\uparrow AUROC$	86.84 ± 2.64	90.12 ± 6.96	50.00 ± 0.0	**94.70 ± 1.75**
$\uparrow AUPR$	94.56 ± 1.00	96.81 ± 2.38	83.33 ± 0.0	**98.23 ± 0.71**

Table 5: Results of multi-class OOD detection methods using PNN. Values are percentages and average across six different out-of-distribution test datasets and four different random seeds, for sequential tasks. The best values are marked in a bold font.

The results of the experiment, averaged over different seeds, are shown in Table 5. We display the mean value and the standard deviation in each metric and for both

loss functions. We know that a lower value of $FPR95$ is preferred, and higher values for both $AUROC$ and $AUPR$. We obtain that the best performance is found with the MaxLogit method and training a Binary Cross Entropy (with logits) loss function: only a 20% of probability for a known sample to be detected as unknown; a 95% of probability of an OOD sample of having a distinguishable anomaly score in relation to an in-distribution sample; and a 98% of average precision. It is interesting to see that we get slightly better results with the Binary Cross Entropy rather than with the regular Cross Entropy. In regards of the MSP, the results with Binary Cross Entropy are not useful: the equation of the anomaly score uses the softmax, so 2 outputs are needed. When using a softmax function over one single output, we always obtain the same output (a 100% score). Therefore, we can understand that the $AUROC$ is 50%, as it is basically giving the same anomaly score to every sample, either it is in- or out-of-distribution.

With this results, we can state that the MaxLogit method is effective for OOD detection, since when computing the anomaly scores for different objects, they can be easily used to discern between samples that are in-distribution and samples that are not. Moreover, we could perfectly use both a Binary or Cross Entropy loss function with the MaxLogits. However, given that our setting is centred on binary classification per head and that the OOD detection works slightly better with the Binary Cross Entropy loss function, we will use this one for the subsequent trainings.

Once we have decided how we are computing the anomaly scores, the next step is to establish how we are using them for OOD detection. A simple but effective way is to compute the anomaly scores of the validation set after training with a given task, so that we have them as reference. Afterwards, when inferring the semantics of new objects, we can compute their anomaly scores too and compare them with the anomaly scores of the trained tasks, so that we can decide if they belong to one of the known categories or not by similarity. As examples, we present in Figure 18 the anomaly scores computed for the validation and test sets of four different categories in the TORO dataset. The histograms display the amount of samples with a given MaxLogit value, and we also plot the mean value (in a dashed colored line) and the σ and 2-σ distribution as the colored areas (darker and lighter, respectively).

We can see in both top panels the anomaly scores for "dark_mat" and "lab_floor" classes. They are examples of well adjusted anomaly scores, that is, both the validation (in blue) and test (in red) samples have more or less the same mean value and they fall within a standard deviation of 2-σ, that is, the 95% of the samples. In these cases, it would be efficient to decide that the anomaly score for a sample to be considered in-distribution must be in the range between the mean of the validation anomaly scores and 2 times the standard deviation:

$$\mu - 2\sigma \leq a \leq \mu + 2\sigma \tag{13}$$

On the other hand, if we look at the bottom panels of Figure 18, we can see two examples where this criterion is not valid. For the "blue_mattress" and "pedestal" categories, this range set by the validation set either does not include all of the test samples, or covers more values than it the expected by the test set. In the first case,

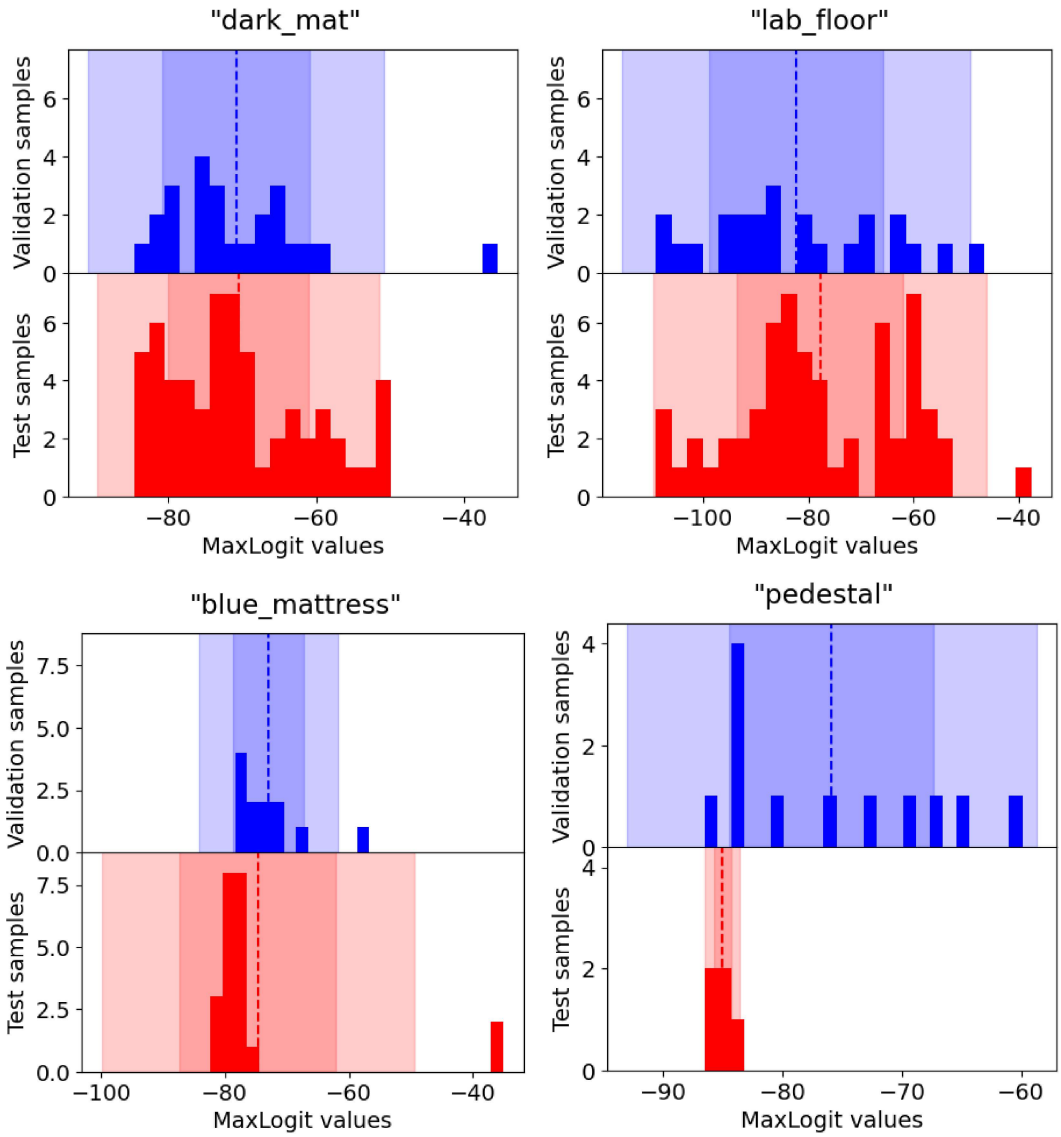


Figure 18: Examples of MaxLogit values for different categories of the dataset. In blue, the values obtained from the validation dataset which are used to set the thresholds for OOD detection; in red the values obtained from the test dataset. The mean of the values in both cases is displayed with a colored dashed line, and the σ and 2-σ thresholds are designated with colored areas – darker color for the σ and lighter color for the 2-σ, representing the 68% and 95% samples of the distributions, respectively.

which is the "blue_mattress" plot, we set a very narrow range for a sample to be detected as in-distribution, so that the probability of a known object to be detected as such decreases. In the other case, which is the "pedestal" plot, the range set by the validation samples is too wide. Therefore, we encounter the other possibility, an OOD object might be detected as known if its anomaly score is negative enough. These problems are results of the variety present in the different sets: the more similar the

validation and the test sets are, the more similar their anomaly scores will be too. However, these scores are different every training, so that it is difficult to take into account every scenario. For this reason, in all of the experiments we will try different random seeds and average the results. For some cases, the anomaly scores might be well adjusted and efficient, for some others it might not, but the general performance is assured to be good.

After taking all the results into account, we then conclude the OOD detection experiment. We are using in the rest of the experiments the MaxLogit method, that takes as the anomaly score the negative maximum of the unnormalised logits returned from the NN. These scores will be computed using the validation set, and we will use a range of $\mu \pm 2\sigma$ to determine if a sample is in-distribution or not. When a new sample is inferred, we will take the unnormalised logits resulting from each one of the trained heads. The minimum of them will be considered the most likely category that the object belong to: if the anomaly score falls within the range set by the validation samples for that head, the object will be considered known and classified as the label corresponding to that head; otherwise, the object will be considered to be unknown.

4.1.3 Backbone and base network

The PNN is a very versatile algorithm, as we are able to decide which classifier is more suitable for our specific application and change it. Consequently, there are multiple valid options in order to set the base NN that will be copied and trained for each new incoming task. Among the most popular networks in CV for image classification, we can find ResNet [24] or Residual Networks. This architecture was conceived to tackle the degradation problem, which occurs when estimating the accuracy with a plain network and the accuracy degrades with the complexity and depth. In this sense, the motivation of ResNet is to make an identity-mapping suitable network, resulting in a NN easier to optimize and gain accuracy from increased depth. It has many variants with the same concept but different number of layers, such as ResNet-34, ResNet-50 or ResNet-101. In Pytorch [43, 53], these models are available and pre-trained with IMAGENET [14], an image database organized according to the WordNet – a large lexical database of English – hierarchy, in which each node is depicted by multiple images. Therefore, we can directly make use of these networks and only train their last few layers. With this, we achieve to take advantage of the ResNet architecture and the pre-training, which helps the network to extract meaningful features, and only modify the last part so that it fits our own dataset. In this way, ResNet would function both as base network, since we would be training at least the last layers for each new task, but also as a backbone in the sense that the frozen layers of the net would serve as a powerful feature extractor.

However, as we described in Section 3.2.1, we want to take a step further and use a foundation model such as DINOv2, able to extract features from unseen datasets and pass them to a simple classifier to obtain their semantic categories. Hence, we will compare the performance of both backbones to see if the decision of using a state-of-the-art tool is actually better than a former consolidated option, such as ResNet. To do so, we will use three different configurations: a ResNet-50 training only the

last layer as classifier, DINOv2 and a single layer NN classifier, and DINOv2 and a multilayer NN classifier. Using the single layer as base network comes from the DINOv2 documentation [41], in which it is stated that the extracted features can be directly used for image classification, with a linear layer applied on the class token and the average of patch tokens. We want to try also a more complex classifier with multiple linear layers that takes as an input all of the extracted features and create a model from them.

For this experiment, we will compare the average accuracy per task (a_{e_i}), the model size per task, and the average training and inference time per task. These metrics will be computed for each one of the three configurations, averaged over different seeds, so that we can compare the performance in the three cases. Additionally, the dataset will consist of the same 6 main terrains with random images every run for each phase. The network will be trained sequentially and each column will be tested with the test sets of all the tasks, whose IDs range from 0 to 5.

In the Figure 19 all the resulting metric values are displayed. The average accuracy per task in the top left panel is computed following the Equation 7, so that only the test results of previously learned tasks are taken into account. It is explicit that in average, the DINOv2 and multilayer NN configuration has the best results, while the other two settings are most likely similar. The exact accuracy result might not be fully representative, as there are many factors that involve this metric, but we can qualitatively extract that the best performance regarding the accuracy will be the one of the foundation model and a multilayer NN.

In the top right panel of Figure 19, we encounter the model size of the configurations per task, that is, measured after training with task i. As we are using PNN, we expect that adding a new column also adds to the size of the network, because we are actually copying the base network as many times as tasks are trained. Consequently, we can see how the ResNet50, which is a heavy model from the start, is the one that occupies more space in the disk, from around 200 MBytes up to more than 600 MBytes after training 6 tasks. The other configurations, using either a single layer or multilayer NN, are not heavier than 100 MBytes after training with all of the available tasks. The multilayer classifier is slighter heavier than the single layer. Moreover, it is important to note that the ResNet-50 is working both as a feature extractor and classifier, while in the other cases we have the DINOv2 functioning as the backbone (it is not copied every new task), and the NN as the base network, which is copied for every column. Therefore, in this case it would make more sense to use a combination of the foundation model and a simple network, as they are much lighter than the full ResNet. As a last note, the model size metric in this case behaves very similar to the GPU usage metric, given that every time the code is run, the full network is moved to the GPU memory. That is another reason why the ResNet-50 might not be suitable for mobile robotics, as its size in disk and its GPU consumption is much higher than its alternative using DINOv2 and a simple NN classifier.

Regarding the training and inference average times per task in the bottom panels (left and right, respectively) of Figure 19, there is not a clear better performance. On the training phase, the single layer classifier appears to take around three times more than the others. This might be explained because, as it is only composed by

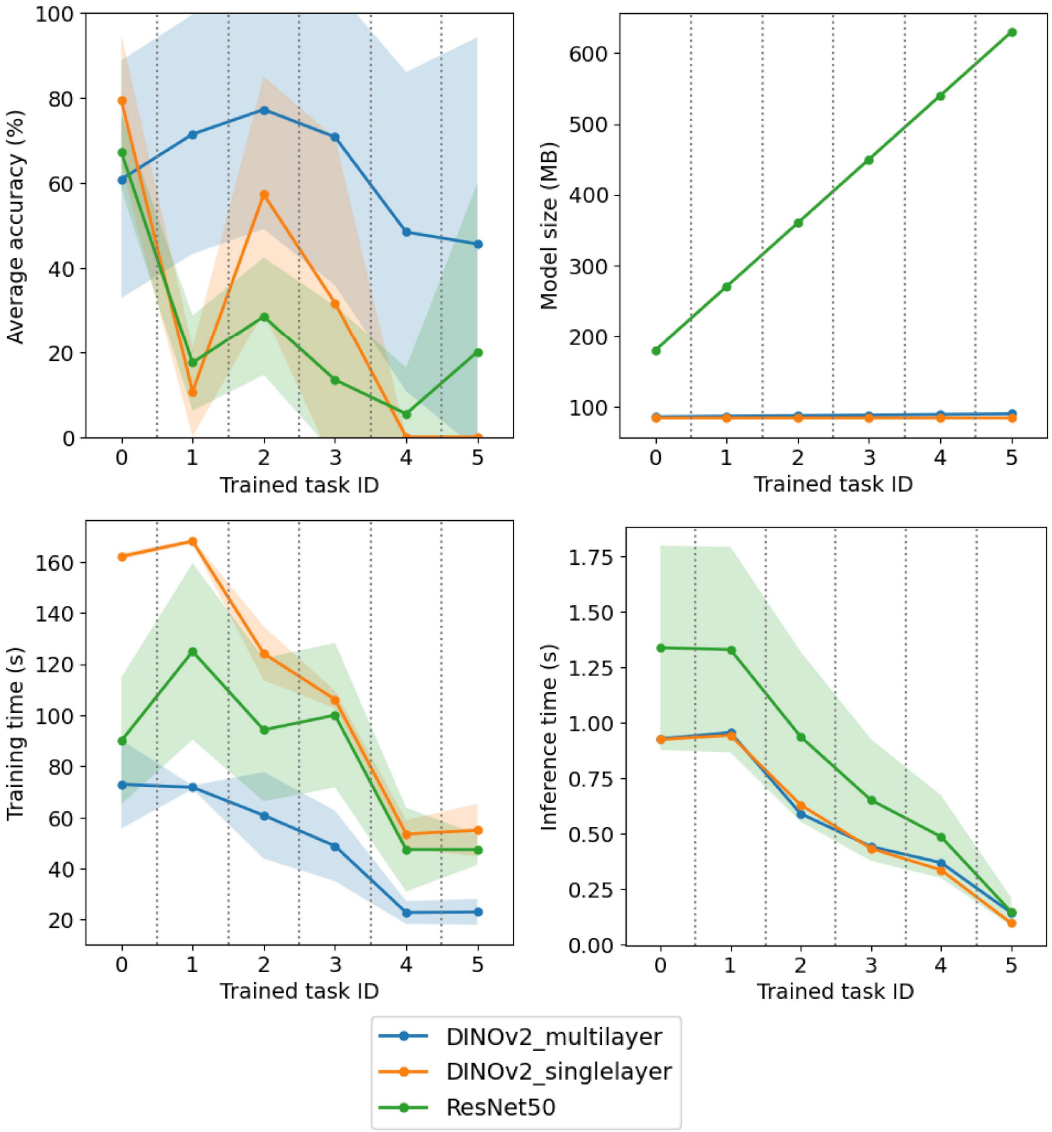


Figure 19: Results of accuracy (top left), model size (top right), and training (middle left) and inference (middle right) time averaged per task for the 3 configurations of backbones and base networks. The legend is in the bottom panel. The mean value across all the tasks is plotted, accompanied by the standard deviation in a lighter shade.

one layer, the loss function takes longer to converge, while in the other cases it can find a local solution much faster. On the inference phase, the difference between the configurations is not notable, taking between 1 and 2 seconds to infer a whole task in the worst cases. For this metrics, we could establish that is not feasible to have a very long training time, in comparison with other set ups, although the training time is still not too long for a real-world application and could be managed in other ways, as we have stated before, e.g. parallelising the training.

Taking into account all the results, we summarised the positive and negative aspects of each configuration in Table 6. While all of the models have a relatively fast

inference, the ResNet-50 seems to be the one occupying more space both in disk and in the GPU memory, and the single layer configuration appears to take longer during training. Additionally, the multilayer setting has a better performance in accuracy terms, so we will continue using it for the rest of experiments. With these results, we have validated the state-of-the-art DINOv2 tool, as we are able to use it in a completely new context with practically no training nor fine-tuning to extract features for images and then use them with a downstream classifier as simple as linear layers. The ResNet-50 has been proven in the recent years to be reliable and accurate, but it might not be suitable in a robotic context where the resources are limited.

	+	−
ResNet-50	Fast training Fast inference	Heavy model
DINOv2 + single layer	Fast inference Light model	Slow training
DINOv2 + multi layer	Fast training Fast inference Light model Best accuracy	None measured

Table 6: Comparison of the three different configurations tested for backbone and base network in PNN, highlighting the positive and negative aspects of each.

4.1.4 Continual Learning performance

After choosing the parameters of the data transforms, the type of OOD method, and the configuration of PNN in terms of backbone and base network, we can fully evaluate the performance of the whole PNN framework with regards to the CL metrics using the dataset collected from the TORO camera. In this way, we have made a series of design decisions that were proven to be the best ones among the other possibilities, but we need to put it all together and evaluate if the performance is cohesive and efficient. To achieve so, we will run the complete PNN with different seeds on the TORO dataset, with random images for the training, validation and testing sets each time, and we will evaluate all the metrics per task: accuracy, GPU consumption, model size, and training and inference time. The accuracy will include both the accuracy per class and the OOD accuracy. Afterwards, we will be able to compute the CL metrics described in Section 3.5.2: the total average accuracy, the average OOD accuracy, and the *BTW* (BTW^+ and REM). With all of these metrics, we will have a complete vision of the PNN working in the TORO dataset and we will be able to complete the whole continual segmentation framework.

Firstly, the class accuracy per task is shown in Figure 20. We plot the mean accuracy of a test task after training with task 1 to task i, and in a lighter color, we represent the standard deviation over different random seeds. It is clear that the accuracy for the tasks before the train set has been trained is null, i.e., for task 2

"lab_floor", the accuracy on the test set after training with the task 1 "dark_mat" is 0, as the category has not been learned yet.

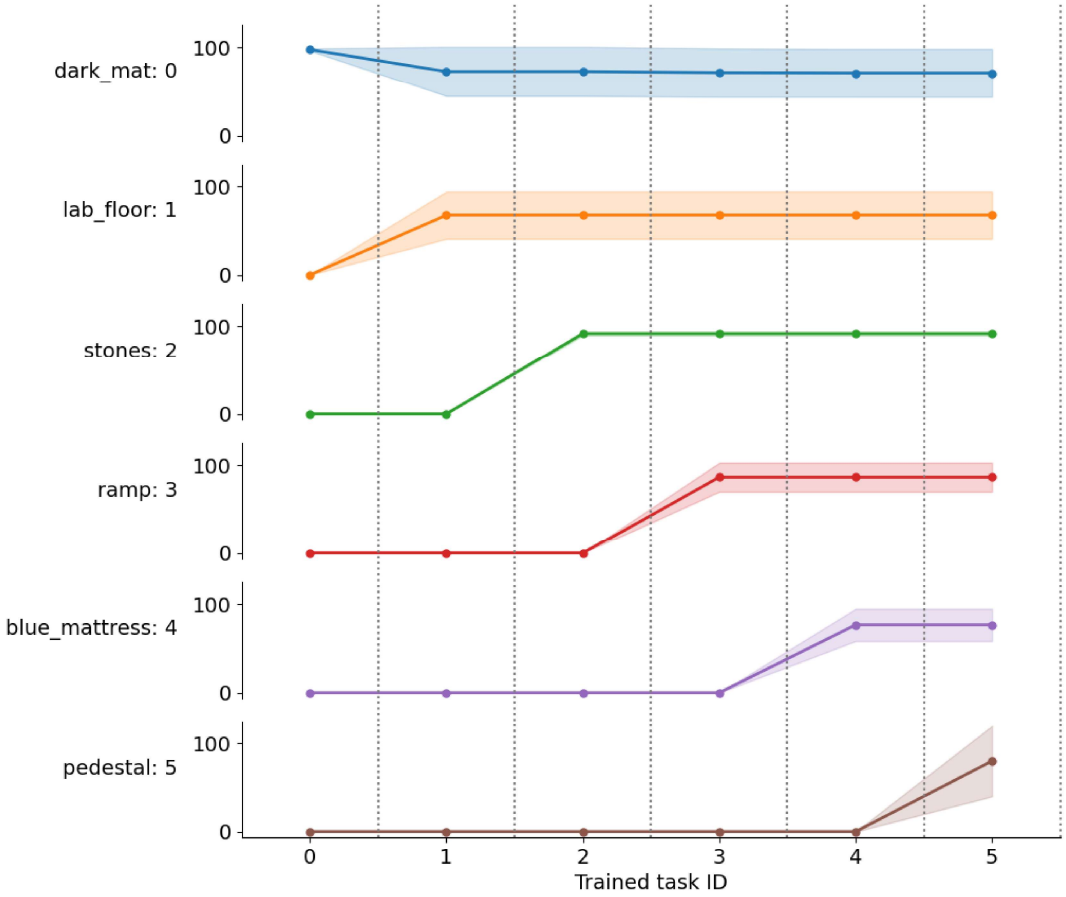


Figure 20: Accuracy results over the training of sequential tasks using PNN. The tasks that have not been trained have an initial null accuracy. The mean values over different seeds is plotted, and the standard deviation is depicted in a lighter shade.

The main objective of using PNN as algorithm is to prevent the catastrophic forgetting, that is, the degradation of accuracy in a certain known task after training with subsequent tasks. Generally, this is achieved, and the accuracy scores are maintained after training with all the tasks. However, we can detect a clear decrease on the "dark_mat" accuracy after training with the second task "lab_floor", from a mean value of 97% score down to 71%. In order to understand why this happens, we must look at images on the dataset belonging to both classes, because there seems to be a similarity between them. In the Figure 15, the differences between both classes are quite notable: the floor of the laboratory is formed by tiny stones tiles, while the dark mat is a grey-ish furry carpet. However, under some circumstances of lightning and due to the relatively low resolution of the TORO camera (640×480 pixels), they might confuse the classifier. As an example, in the Figure 21 we present an example of both categories – "dark_mat" on the top left and "lab_floor" on the top right. In this case, they are visually very similar, almost indistinguishable even for a human

operator. Therefore, we may think that the decrease in accuracy in the "dark_mat" class after training with "lab_floor" is justified, and it mainly depends on the similarity between the images on the training sets of both categories.

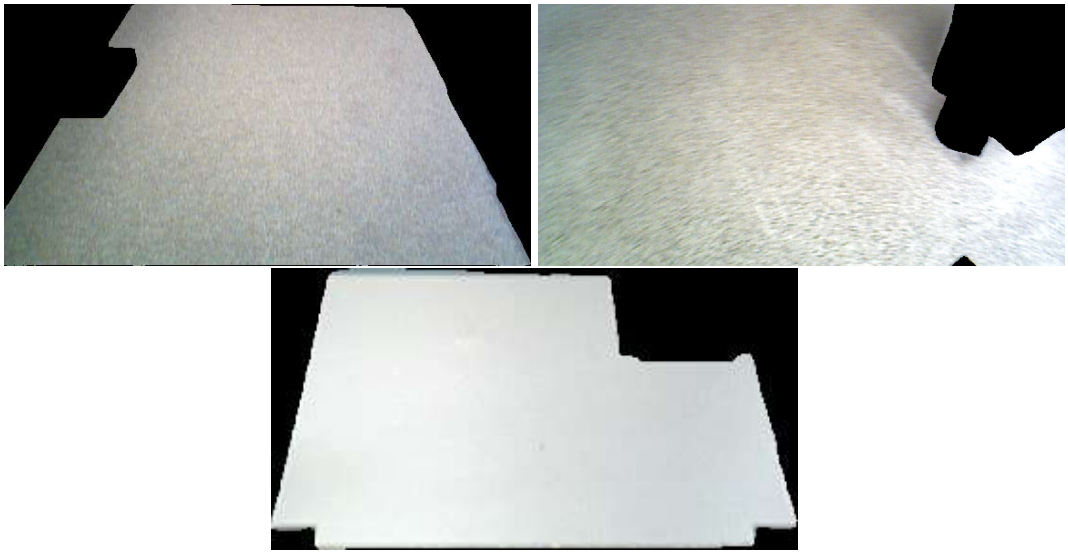


Figure 21: Examples of images belonging to "dark_mat" (top left), "lab_floor" (top right), and "ramp" (bottom) classes. Terrains belonging to different categories might be visually similar under certain lighting circumstances, or when the robot makes a turn and the captured frames become blurry.

Regarding the rest of categories, we encounter accuracy scores ranging from around 75% to 92%. The total average accuracy, OOD accuracy, and Backward Transfer are shown in Table 7. The average accuracy is notable, around 79%, which validates the power of the classification in CL. Additionally, there are no signs of remarkable catastrophic forgetting, as the BWT is null and the REM is a full 100%; the only remarkable decrease in accuracy is explained by similar samples from different categories on the dataset. As expected, the BWT^+ is also null: the algorithm does not have knowledge of unknown objects before training a head for them, so the transfer of knowledge in a backwards way is not contemplated. Regarding the accuracy of the OOD samples, it is around a 70%. The worst OOD accuracy values are found in the "lab_floor" and in the "ramp" categories. Looking back at the examples in Figure 21, those classes are very similar between each other. If we take into account that the first task that is trained in all the experiments is "dark_mat", it might be understandable that both "lab_floor" and "ramp" tasks are wrongly labelled as "dark_mat" instead of unknown terrains. The complete accuracy matrices for both the in-distribution and OOD sets are shown in Annex B.

Moving on to the metrics related to computational resources, in Figure 22 we depict both the model size and the number of parameters. They increase linearly with every task, because every time we train a new head the base network is copied and the number of parameters that it has is added. The initial PNN has around 85.95 MBytes, and for each task, it is increased in 0.834 MBytes. Hence, after six tasks

A	A_{OOD}	BTW	BTW^+	REM
78.86	70.78	0.00	0.00	100.0

Table 7: Average class accuracy, average OOD accuracy and Backward Transfer values for the PNN method.

the weight of the network is around 90 MBytes. The same behaviour is presented in the number of parameters, going from around 434 000 at the beginning up to 1.5 million parameters after training all the tasks, with an increase of 217 041 parameters after adding a new column. Regarding the GPU memory consumption, it might vary significantly depending on different factors in the training, such as the number of images in the datasets or the number of columns in the PNN, but in average, this experiment consumed between 800 MBytes and 1 GByte of GPU. These are not high values, but we must keep them in mind when transferring the method to a mobile robot with limited resources.

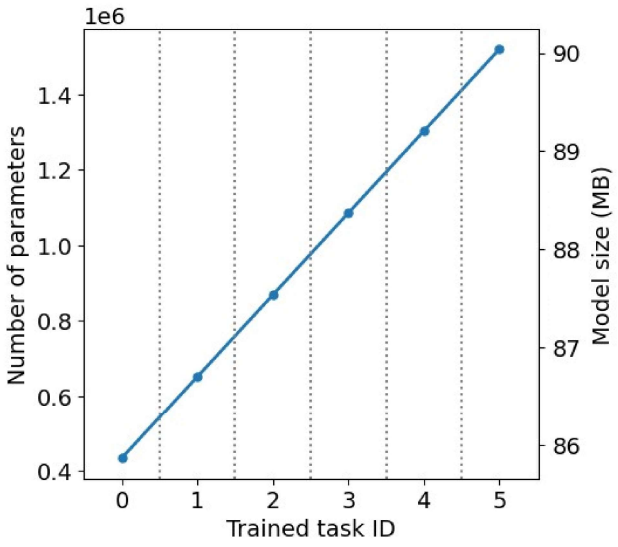


Figure 22: Number of parameters and model size variations in function of the trained tasks in PNN.

Finally, we expose the training and inference average times per trained task in Figure 23, left and right panels, respectively. They depend mainly on the number of images that the training and test set have, so that is why we observe a variation in the times between tasks. In the case of the training time, it can vary from around 25 seconds up to 75 in the worst case. It might also depend on the randomness of the training process, as sometimes it takes a little bit more of time to find the convergence of the loss function. Looking at the inference times per task, they are the same for a certain test set after training with every train set, ranging from a few tens of milliseconds up to 1 second.

All these metrics support and validate the correct working of the PNN in the way that it was designed. The training of different heads for different classes and using

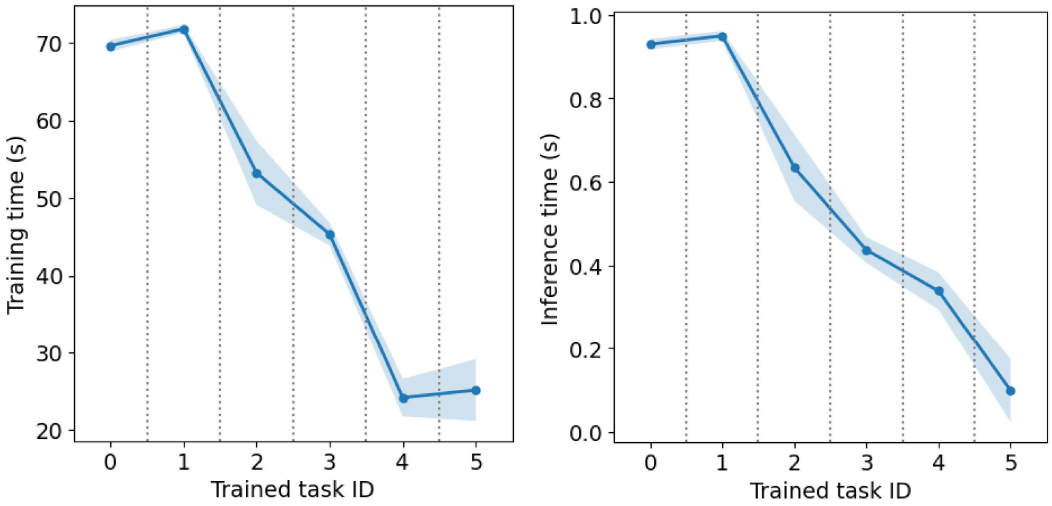


Figure 23: Average training time per task (left panel) and inference time per task (right panel) in seconds. The variation between tasks correspond to the number of images that the training and test sets have.

an OOD detection method enables a correct classification of 75% of the categories, and 70% of the unknown objects are detected as such. The most probable reason for these scores is that the dataset contains classes with images very similar visually, so that even for a human they would be difficult to distinguish. In relation to the computational resources, training a new object adds less than a MByte and the times for training and inference are quite acceptable for an online setting. All in all, we think that the performance of the PNN is notable and we accomplish to complete a method for task-agnostic CL classification.

4.1.5 Progressive Neural Networks robustness

The last experiment in this set verses on the ability of the PNN to be robust, independently of the number of tasks that it is trained on. In theory, the CL network does not have a practical limit in learning terms, that is, we could potentially train infinite tasks and the PNN would still work correctly. However, this might not be a realistic experience, so it would be interesting to see the behaviour of the algorithm. Therefore, we will run an experiment in which we will create pseudo-tasks formed by one image generated randomly, and we will train the PNN with them. We will do so for 200 tasks and we will evaluate the model size, GPU usage, and time metrics. We will not take into account the accuracy score in this case nor the CL metrics, because the classification power of the method has already been evaluated using a real dataset in Section 4.1.4. Instead, we will analyse how the PNN manages the computational resources when training so many heads, to materialise if it is actually feasible to have a large CL classifier.

In this regards, we display in Figure 24 the mentioned metrics. The model size, in the top left panel, follows the same trend that the one exposed in Figure 22, but

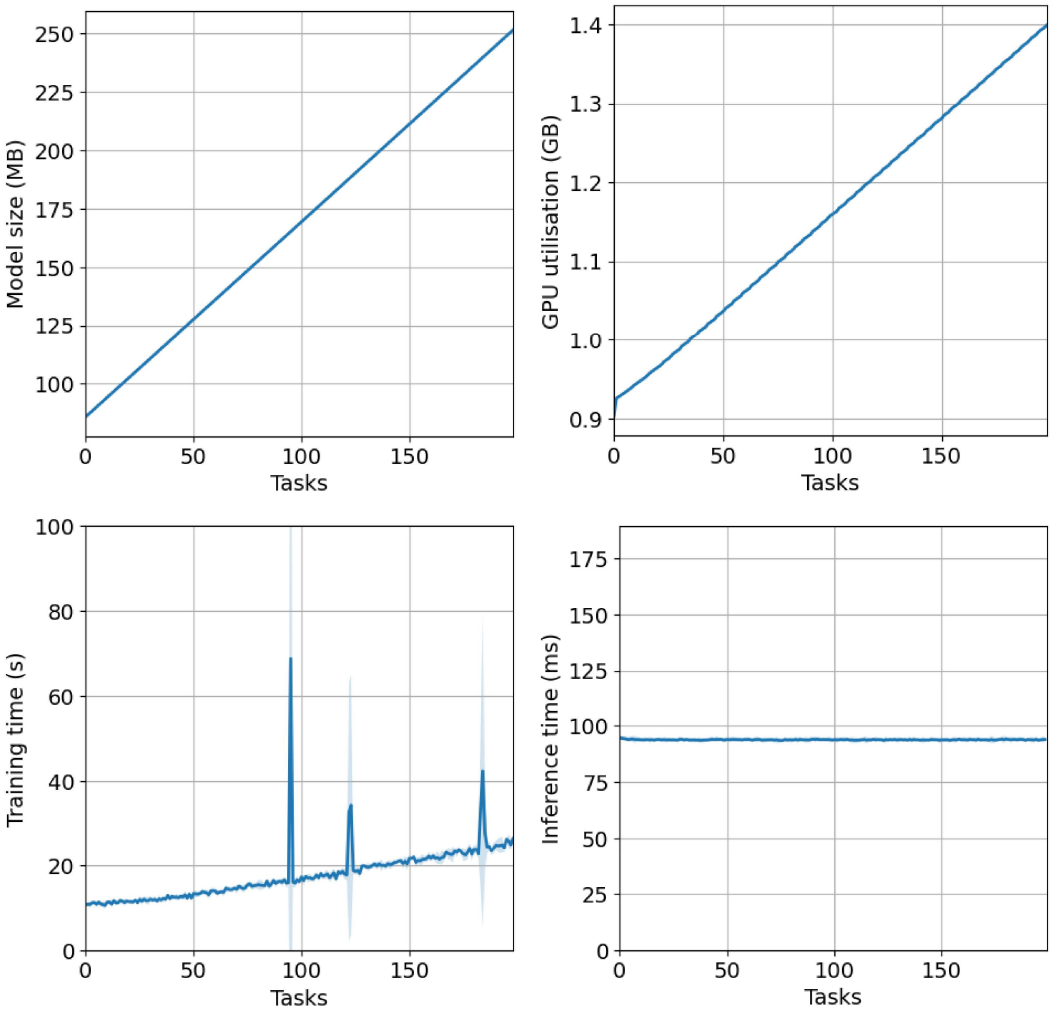


Figure 24: Results of the robustness experiment using PNN: model size per task (top left), GPU utilisation per task (top right), and training (bottom left) and inference (bottom right) times per task.

it encompasses a larger number of tasks. Therefore, the initial model size and the increase in size per task are the same in both figures, and we can check how for 200 tasks, the model occupies around 250 MBytes. We consider that it is still a relatively low number, and almost every mobile robotic system nowadays would be able to store it. In terms of the GPU consumption, in the top right panel of the Figure 24, it also increases linearly. However, it is worth noting that the amount of size occupied after training a new task is not the same as the size of a new column. While a new column is around 0.8 MBytes, the slope of the GPU utilisation plot is around 2.5 MBytes. Therefore, after the 200 tasks, the method utilises from 900 MBytes of GPU, up to almost 1.4 GBytes, an increase of around 0.5 GBytes. In terms of computational resources, it is a huge change and we have to be very careful with this, as most of the times the amount of available GPU memory is a limiting factor in a robotic system.

Concerning the algorithm times, they are displayed in the bottom panels of

Figure 24. The inference time (bottom left panel) is roughly 0.1 seconds in all the tasks, given that the test set only contains one image. However, the average training time shows an increasing trend. The train set also contains a single image, so it is interesting to see how the more images the set has, the more time it takes to train. Nevertheless, the difference is very low, of below 10 seconds.

Taking all the results into account, we can conclude that it is possible to train a very large working PNN classifier. The model size increases linearly with the number of tasks, with a slope of 0.834 MBytes per task. Even though we are using a parameter isolation CL method, the extra heads that we train with new tasks are not that heavy and any mobile robotic system could potentially be trained with hundreds of tasks with no problem. In terms of the GPU, we encounter the same behaviour but it might be a limiting factor as the memory size that the network occupies increases in larger steps than the model size itself. The inference average times are kept low in all the cases, but the training times increase with the number of tasks. At least for a training set consisting on one image, the variation in training time is not that remarkable, but it should be a characteristic to take into account. Lastly, although we have not analysed the accuracy of this experiment because we trained and tested with randomly generated images, it is worth mentioning that we think that it might be the most limiting factor of all. When training some tasks, we use the validation set to compute the in-distribution anomaly scores and decide thresholds for identifying OOD samples. In Section 4.1.4, we could see how some classes were confused with each other and OOD samples were classified as known objects, because of bad lightning in some frames or quick camera movements. In a huge dataset with thousand of different categories, for example, it might be complex to have very visually different classes so that they do not look similar under any circumstance. This might be a disadvantage of using the algorithm with large datasets, but further work has to be done in this side.

4.2 Video sequences experiments

Once we have validated the performance and the classification power of the PNN, the CL part of the method, we can join the segmentation tool and the learning network in the same pipeline. Consequently, we have a complete method that puts together SAM-Track, a powerful segmentor working as a foundation model that does not require previous training and is aided by an efficient video tracker; and task-agnostic PNN, a CL method that is able to classify the known objects in specific categories, and to detect the unknown ones and train a new classification head for them.

As described in Section 3.4, the dataset collected using the TORO robot has 7 different annotated video sequences. Therefore, this experimentation part will focus on running the full method on them. For that purpose we will firstly train a PNN network wit random images belonging to the main walkable terrains and appearing in other sequences different from the one that is being evaluated in each moment. With the validation set, we will compute the in-distribution anomaly scores of the trained categories, and we will suppose that the rest of objects of the dataset are considered as unknown. In this case we do not create a test set, as we will be evaluating the output of the segmentation tool for each frame.

Thus, we will obtain two groups of results: quantitative results of the classification performance of both known and unknown objects, and qualitative results of the both the classification and the segmentation results. In this sense, we will expose in the first place the accuracy scores for the different sequences and the different categories in the dataset, and then we will compare some segmented and classified frames with correct and wrong results. With this last set of experiments, out objective is to complete the evaluation of the full locomotive perceptive method and ensure its correct working in a real-world setting.

4.2.1 Quantitative results

In the first place, we will be able to compare the classification of the object in the center point prompt in each frame to the annotation. We can obtain then an accuracy score per class and a total average accuracy, depending on the number of correct classifications among the total number of objects. We ran the method on the different 7 frame sequences, with random seeds and training with different combinations of images per class, avoiding objects belonging to the sequence that we are evaluating.

When starting the method, the followed procedure is described in Figure 14. The point prompt is located in the centre of the frame, that is, in $(x, y) = (320, 240)$ pixels, and the algorithm will always segment the object that contains the point prompt. Afterwards, the classifier is run and a semantic label added to the object. If it is identified as "Unknown", PNN trains a new column using the mask of the object and learn its label, which is assigned by a human input. In the next frame, if the object still contains the point prompt, the tracker is run and the same semantic label is kept; otherwise the segmentation is run, and the described process is repeated.

At the end of the sequence, we have a dictionary with the label assigned to the object containing the point prompt at every frame. This dictionary will be compared with the annotations and we obtain the results in Table 8. There, we expose the accuracy score of every category appearing in each sequence. We also compute the average accuracy of the known objects (pre-trained known terrains) and the average accuracy of the unknown objects (not pre-trained, learned online), plus the total average accuracy at the bottom for each sequence. It is worth mentioning that the classes are not balanced: some objects might appear in more frames than others.

There are multiple conclusions that can be drawn from the results obtained segmenting and classifying the video sequences. Starting with the known terrains, we encounter average accuracy scores from around 71% up to 93%, with an average accuracy among all the sequences of 75.54%. These are notable results, and we can deduce that the algorithm does not suffer from catastrophic forgetting. Even if we are including new unknown objects to the PNN, the accuracy of the known object is high in average and we can consider that the algorithm works efficiently as a simple classifier. Among the different results, most of them are notable and expose the good performance in classification terms. We find 3 extremely low scores: a 0% for "blue_mattress" in sequence 2; a 41.18% for "dark_mat" and a 40% for "stones", both in sequence 3.

The first case corresponds to a segmentation error, which we will comment in

	Sequences						
	1	2	3	4	5	6	7
dark_mat	73.78	80.29	41.18	86.32	80.52	63.42	81.67
lab_floor	100.00	93.08	96.55	95.49	95.35	50.85	74.22
stones	85.71	82.50	40.00	82.86	98.00	94.44	85.71
ramp	-	98.46	-	93.75	99.09	100.00	89.41
blue_mattress	-	0.00	89.45	-	-	100.00	66.67
pedestal	-	-	100.00	-	-	99.51	90.00
Avg. prior categories	91.87	87.14	77.74	93.09	85.07	71.14	76.15
wall	100.00	73.33	66.67	20.00	-	-	75.29
PC_table	-	96.97	-	99.29	-	-	67.35
plane_wall	-	100.00	85.71	89.57	-	-	-
extinguisher	100.00	86.67	-	100.00	-	-	82.61
cable	-	87.50	-	66.67	-	-	66.67
TV_mount	100.00	100.00	100.00	-	-	-	100.00
bistro_table	-	98.75	-	-	-	-	51.54
person	-	-	-	-	100.00	100.00	80.00
stand_board	-	90.00	-	80.00	-	-	-
Avg. learned categories	100.00	92.66	85.71	78.86	100.00	100.00	68.33
Total avg.	**92.25**	**89.21**	**78.62**	**87.57**	**85.45**	**71.21**	**75.06**

Table 8: Accuracy percentage for each category of the pre-learned and unknown objects for every sequence collected from the TORO camera. The average accuracy for both prior and newly learned categories is also shown, and at the bottom, the total average accuracy per sequence. Note that the classes are not balanced.

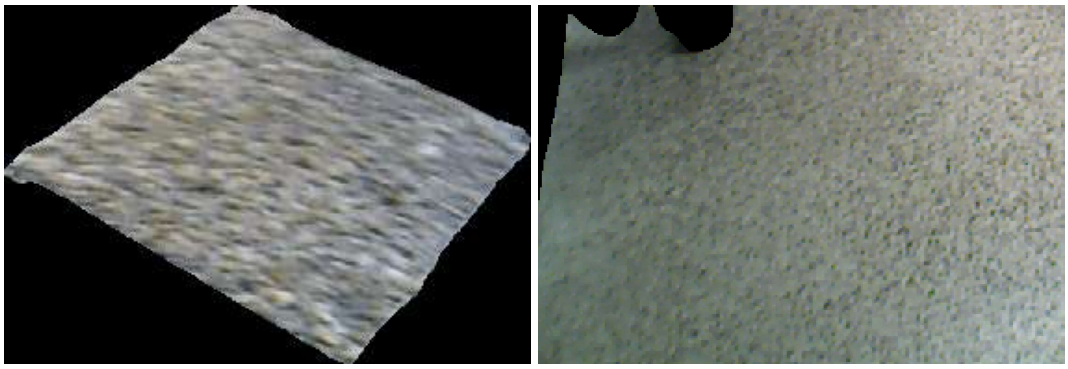


Figure 25: Cropped objects detected in frames of the sequence 3, belonging to "stones" (left) and a close-up of "lab_floor" (right) categories.

Section 4.2.2. The second and the third ones correspond to classification errors. In both cases most of the times the terrains were mistakenly labelled as other category, also other known terrains. The "dark_mat" was confused with "lab_floor" and "ramp";

this recurrent error has already been described in Section 4.1.4. On the other hand, the "stones" were identified as "lab_floor". Looking at the examples in Figure 25, we see the segmented objects belonging to sequence 3: "stones" in the left panel and a close-up of "lab_floor" in the right one. We train the PNN with cropped images with a 280 pixels resolution, and in these cases the pictures might look similar. Moreover, the "stones" are first detected when the robot is performing a turn, so that the image received from the camera is blurry. Taking into account that the "lab_floor" is described as polished-stones tiles, the confusion between the two categories for this particular case might be understandable.

Moving on to the unknown and learned objects, the average accuracy scores range from 68% up to 100%, with an average accuracy among all the sequences of 84.75%. For the newly learned objects we can find multiple perfect accuracy scores, which is because we are explicitly indicating the semantic information of the object in the frame, and that object might or might not appear again in other frame of the sequence. We only identify a very bad score, of 20% in the "wall" category of sequence 4. This category is specially confusing, as in the laboratory there are two different walls. Moreover, depending on the segmentation tool, different parts of the wall might be segmented, so it is complex to cut the object and also to classify it. In Figure 26 we display two example frames where the object "wall" was segmented. We can see how the segmentation covers different parts of the object in each case, and even if the pictures look similar, the visual information is very little. A human with no other context except from these frames would have difficulties to identify them as walls, so it happens similarly for the method.

Figure 26: Cropped objects detected in frames of the sequence 4, belonging to "wall".

Apart from the mentioned case, in the Table 8 it is remarkable how well the algorithm is able to detect unknown objects and learn them. Most of the accuracy scores are notably high in the case of the learned categories, which demonstrates the ability of the method to identify OOD samples. Lastly, we can find in the bottom row the total average accuracy score for each one of the frame sequences. The values range from 71% to 92%, which are great scores that validate the classification power of the algorithm. They mean that most of the frames were correctly classified, and the added semantic information was right. In the case of the known objects, we can deduce that training with prior samples that might be similar to the objects in the

sequence that is being evaluated produce generally very good results. We must take into account that the sequences, even if they are not completely equal among them, they are similar enough so that the objects detected in them are also detected in other sequences. In regards of the unknown objects, the chosen OOD detection method has a good performance, and training with just one or two samples the new categories is not a problem for posterior classifications, as the accuracy scores are typically high.

We have encountered two other relevant problems during the execution of the method. One of them is related to the OOD detection, even though it has generally a good performance, it also fails from time to time. When detecting unknown objects, some of them might be complex, e.g. "wall" exemplified in Figure 26, "bistro_table" or "PC_table". We expose example frames showing the last two objects in Figure 27. The "PC_table" is a table where the PCs that operate the robot are, so we can find a lot of different components: cables, Wi-Fi router, screens, mouses, DLR logo, etc. It is not only complex to segment it, but also to classify it as a whole single object. It happens the same for "bistro_table", which can be differentiated in three parts: metallic base, middle part and wooden top. In these cases, it might be necessary to learn them multiple times, depending on which part the segmentation has identified. Therefore, even if we encountered one of these objects in a previous frame, it might be detected as unknown in a subsequent frame because the segmented part is new. This can be solved creating new and more specific classes, that is, dividing the complex objects in multiple, simpler ones.

In the same line, this can also happen with the pre-trained objects: an in-distribution sample is detected as OOD. In this case, it is difficult to proceed: training a new head might be a waste of time and computational resources, as the object has already be learned. On the other hand, we can see it as an opportunity to expand a previously learned class and add new information for a known category. This area of work, training multiple heads for the same category, could be explored in future research to see the advantages and disadvantages of doing so. In this thesis and during the experiments, this situation has been presented in some frames, and we have proceed to learn a new head for the category instead of skipping it for simplicity purposes.

Figure 27: Frames containing the "PC_table" (left) and "bistro_table" objects.

Continuing with the accuracy scores, we can also compute the confusion matrix

of the classification for this part of the experimentation with the video sequences. To calculate it, we took into account the true label and prediction of each frame for different seeds in all of the 7 available sequences. The matrix is displayed in Figure 28 and, together with Table 8, they are the summary of the classification performance of the proposed method. The true labels are presented in the y-axis, while the predicted labels are in the x-axis. It is normalised by the total number of objects per category, and the diagonal presents the percentage of correct classifications per class. In a blue rectangle, we enclose the confusion matrix of the known and pre-trained objects, while the complete matrix is the extended version that also includes the unknown and newly learned objects.

The elements in the diagonal have high values, over 90% in most of the cases, demonstrating a good classification performance of the PNN in almost every category. There are 3 lower scores, corresponding to "dark_mat", "wall" and "bistro_table". These categories and the issues in their classification have already been discussed in this Section 4.2.1. Therefore, we can generally deduce that the classification using PNN and the configuration of DINOv2 and a multilayer NN classifier results in a very notable performance, which is evident in the final confusion matrix.

Lastly, it is important to also have into account the computational resources consumed in this part of the experimentation. Regarding the GPU usage, the full method including the segmentation and the classification parts started occupying around 3 GBytes of GPU memory. With the increase of tasks, the model was also occupying more and more GPU memory, similarly to what is seen in the top right plot of Figure 24, up to around 6 GBytes in the sequence with more frames. This quantity also depends on the models that we are using for each one of the parts of the method. We described in Section 3.1.2 the models employed in the experimentation, which were the heavier ones for segmentation and tracking, so that there is margin for reducing the GPU usage in a real application. Regarding the training times, since the training set of unknown categories only contain one sample (the current segmentation), it was generally fast and did not last more than 30 seconds in the worst case. The inference time was not high either, of tens of milliseconds. The method generally can process in an online manner up to 10 fps, it is not extremely fast but it is enough to be aware of the surroundings and perform segmentation and classification of the objects. This value can also be alleviated by using lighter models of the segmentation and tracking parts of the algorithm.

4.2.2 Qualitative results

The classification power of the whole method has been validated using statistical values in the previous Section 4.2.1. However, there is little information on the visual nature of the results that the method returns. For that reason, this Section will delve into the qualitative aspects of the segmentation and the classification of the frames of the video sequences recorded using the TORO camera.

Even though we are not exploring the segmentation results and the tool itself, it is worth looking at the segmented and classified frames. In the following pages, different frames are shown portraying a variety of results obtained using the proposed method.

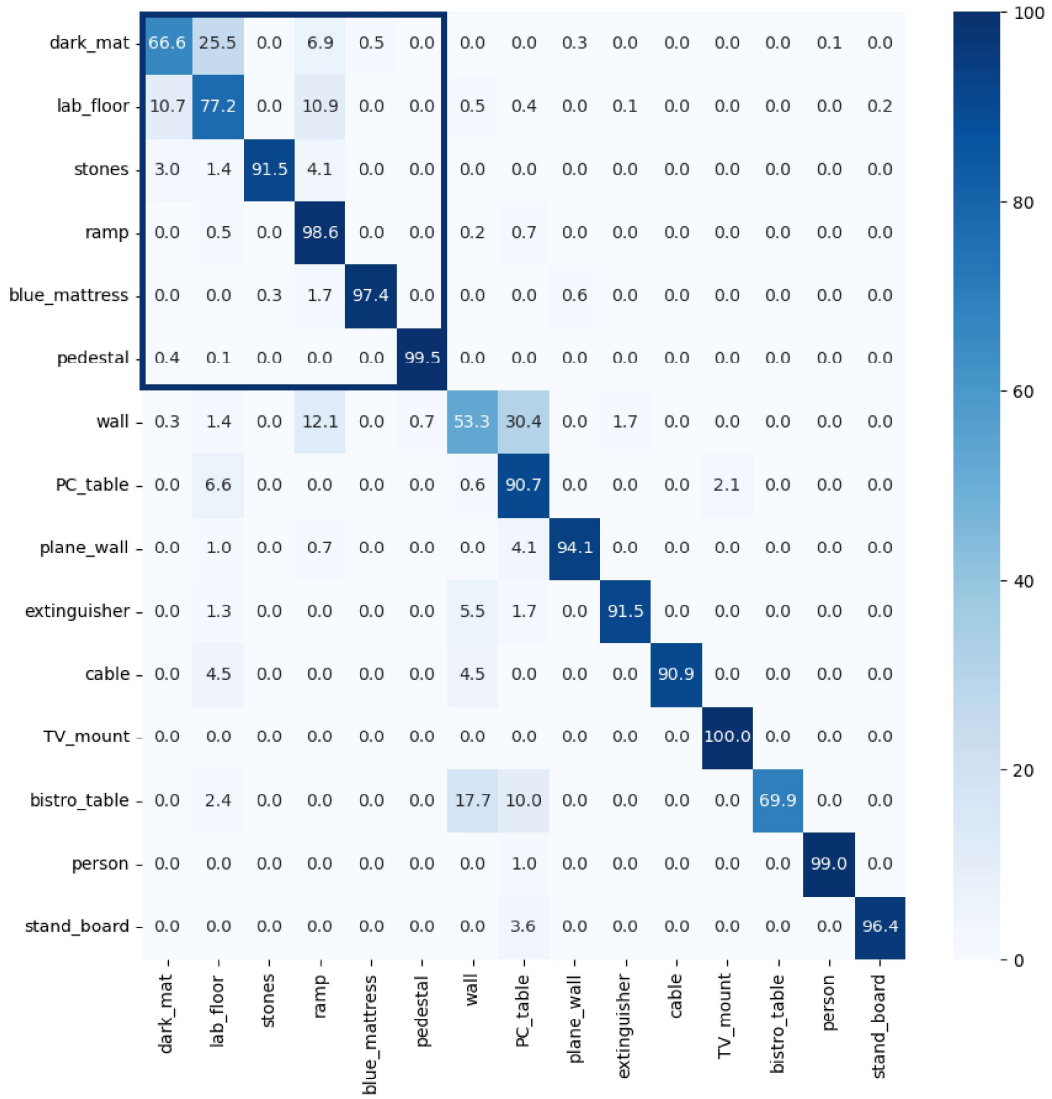


Figure 28: Confusion matrix representing the percentage of objects belonging to a category i (true label) and predicted to be in group j (predicted label). The diagonal represents the percentage of correct predictions per category. The confusion matrix of the known pre-trained categories is enclosed in a blue rectangle, including from "dark_mat" to "pedestal".

There are 4 groups: correct known and unknown objects, and incorrect known and unknown objects. For comparison, the original frame is shown in the left panel, while the resulting frame from the method is in the right panel. In order to visually distinguish different outcomes, we use three colors in the output frame:

- Green: object is segmented and a known semantic label is added.
- Red: object is segmented but it is unknown.
- Blue: object is not segmented.

Since we are using the point prompt mode and CL, that is, we are learning the new objects directly when they are detected, we only have either green or blue objects: green for the object containing the centre point prompt, and blue for the rest of the frame. Other variations, such as using the method without learning, might include results with "Unknown" objects and red colors.

Looking at the first results in Figure 29, four pairs of frames are displayed with correct segmentation and classification of the objects in the centre point prompt. The identified object is contoured and colored in green, while the rest of the frame is shaded with a blue color. In the middle of the frame, where the point prompt is located, a blue cross is drawn. The semantic label of the object is written above the cross in the corresponding color and with a white background.

The next group of frames in Figure 30 show incorrect segmentation and classification of known categories. In some cases, the mistakes are caused by a rapid movement of the camera, so the segmentation is not good. Therefore, the classification is not good either, so the first mistake is propagated through all the algorithm.

In Figure 31, there are frames containing a correct segmentation and classification of previously unknown objects, that are learned online. In these cases, the contours of the objects are more complex than the main terrains, but the algorithm manages to identify them and assign the correct label.

Finally, in the Figure 32 we display examples of unknown objects whose segmentation and classification are not correct. These cases were quite common, as mentioned before, the unknown objects can have complex shapes that the segmentor is not always able to detect correctly. Therefore, the classifier receives as input a distorted image, and if it contains traces from other categories, it might be mistakenly assigned to them. For instance, in the examples we have "wall" in the first pair of frames and "bistro_table" in the third pair of frames, but they are not segmented nor classified correctly because of the AprilTag. In the first case, it is detected as "pedestal", which also has a tag on it, and in the third case it happens the same but with the "wall" category, which also has an AprilTag.

These examples prove the good performance of the algorithm. On one side, we validate the segmentation and tracking tool, a foundation model that we have not trained in order to make it work. Even though it is not perfect, the simplicity of the algorithm to return more than acceptable results is impressive. On the other side we have the CL method, that is able to receive the segmented images and assign semantic information to them. This is all done in the same work pipeline, hence we have designed and validated a perceptive method for continual semantic segmentation in a mobile robotic systems.

More examples of segmentation and classification of consecutive frames are shown in Annex D.

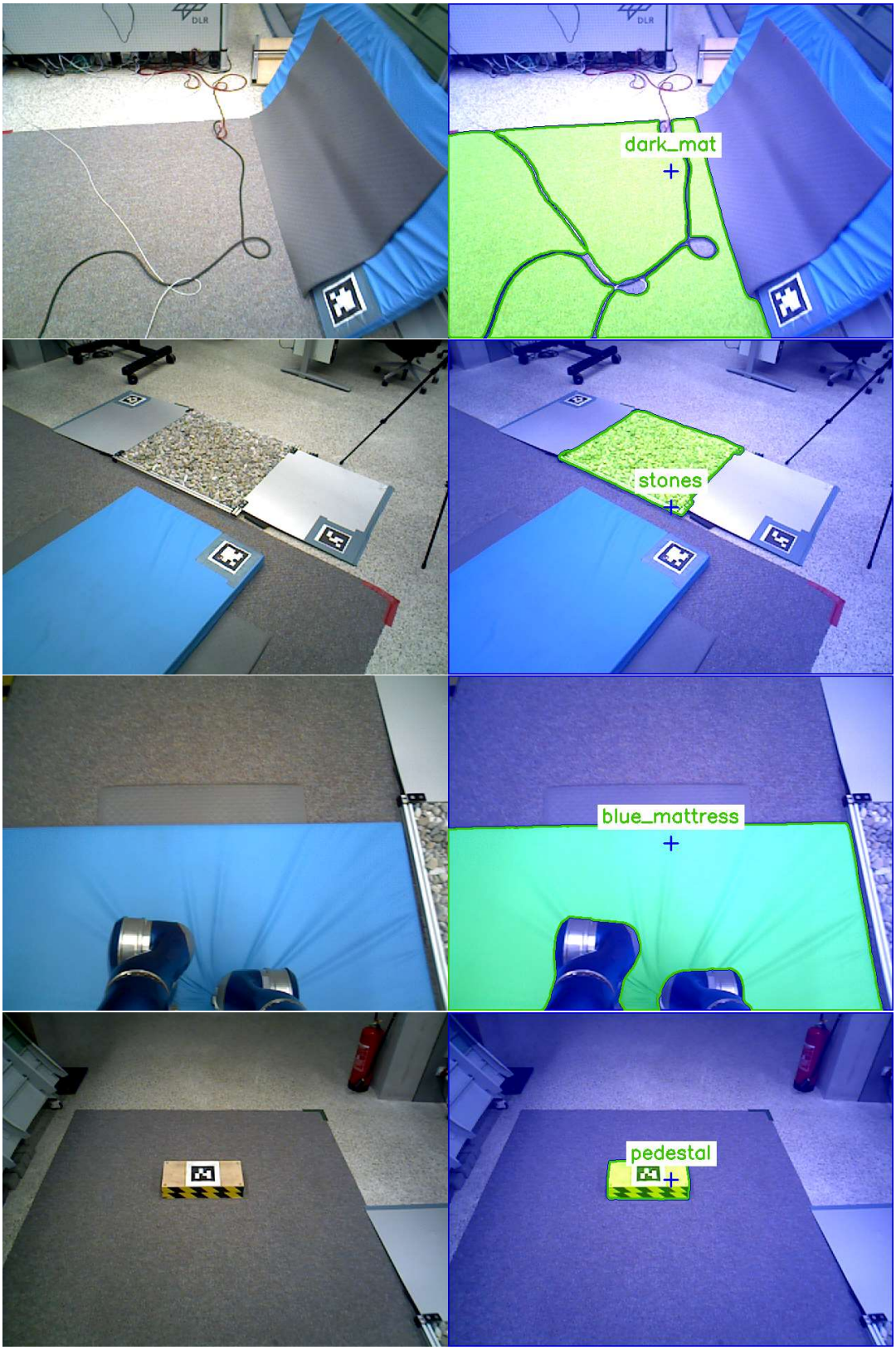


Figure 29: Examples of correct segmented and classified frames with the known and pre-trained terrains.

Figure 30: Examples of incorrectly segmented and classified frames with the known and pre-trained terrains. The correct labels would be, in order from top to bottom: "dark_mat", "lab_floor", "stones", and "blue_mattress".

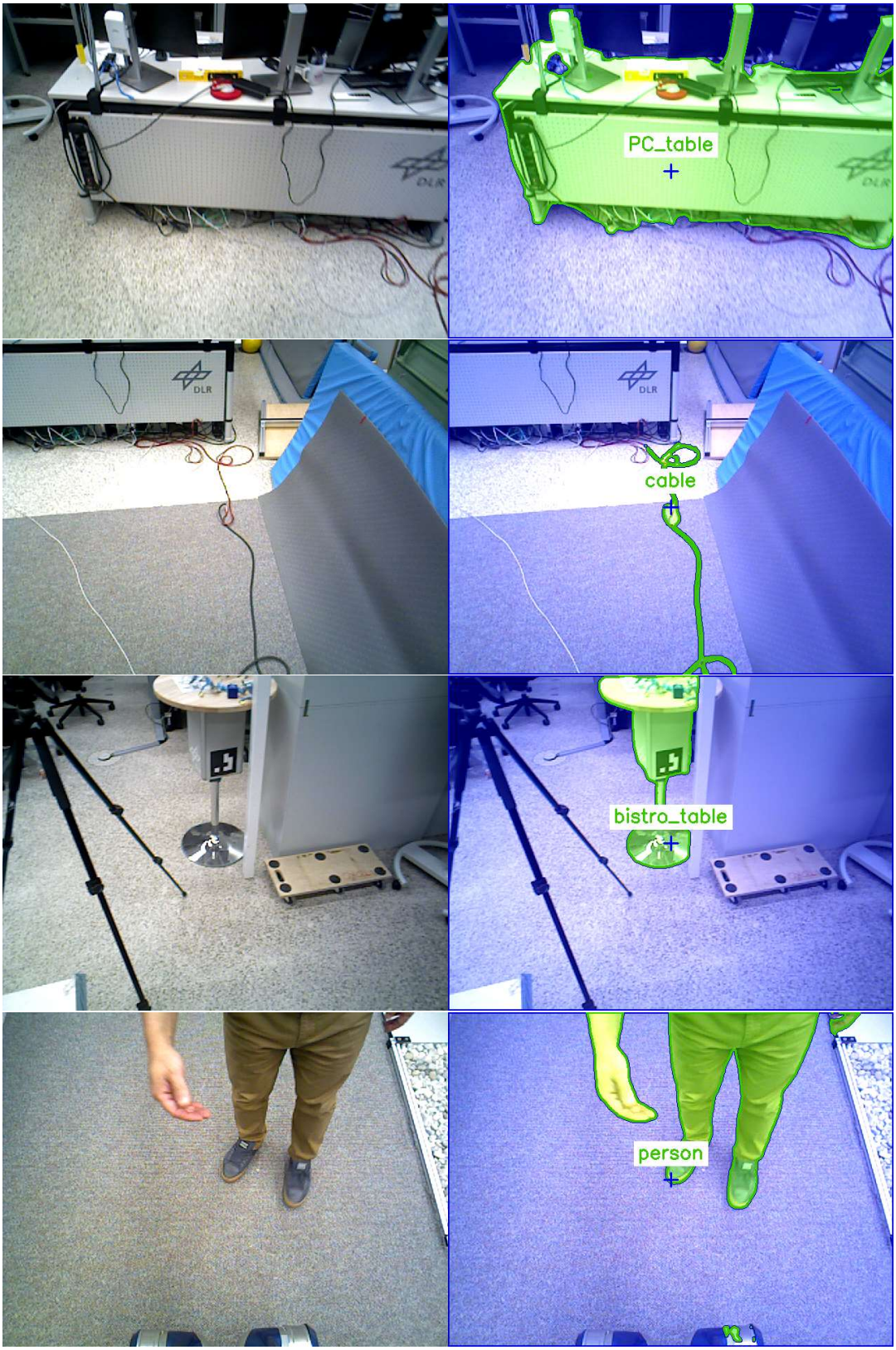


Figure 31: Examples of correct segmented and classified frames with the unknown objects.

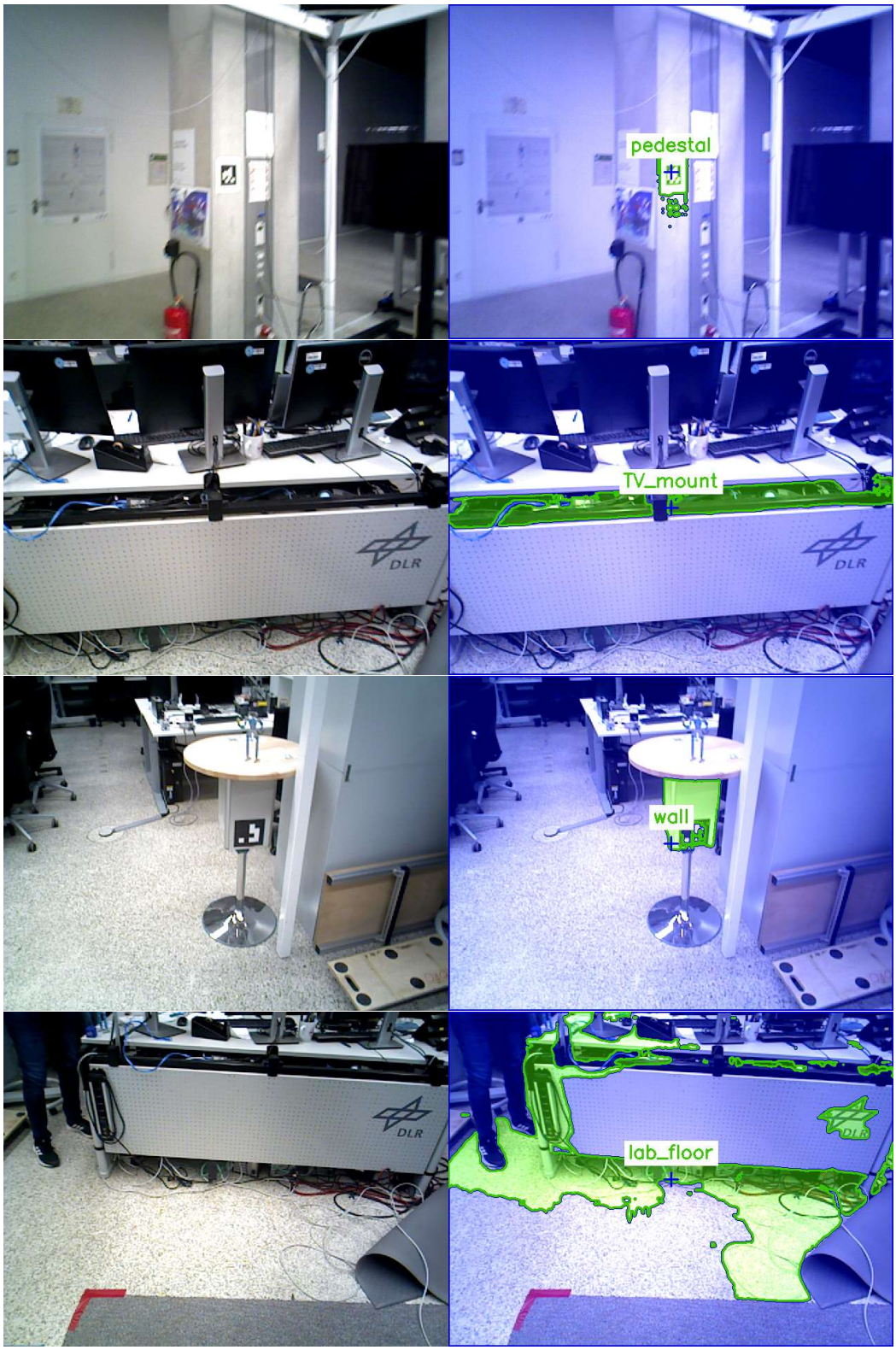


Figure 32: Examples of incorrectly segmented and classified frames with the unknown objects. The correct labels would be, in order from top to bottom: "wall", "PC_table", "bistro_table", and "cable".

5 Conclusions and future work

In this thesis, we designed and integrated a Video Object Segmentation tool with a Continual Learning (CL) algorithm, adding the semantics to the resulting objects and also learning from the unknown. The aim of the method was to be used in a vision system for mobile robotics, such as the TORO robot developed in Deutsches Zentrum für Luft- und Raumfahrt. Having TORO in mind as a possible application of the proposed method, we collected a dataset of the different terrains and objects that the robot encounter in its walking sessions and that we could use for our experimentation. It consists of six main walkable terrains and other nine objects that are visible by the robot under some circumstances, and seven different video sequences as seen from the TORO camera with a total of around 10 minutes of video.

In relation to our method, we started with two independent methods: Segment and Track Anything for image segmentation, and Progressive Neural Networks for image classification and lifelong learning. Our focus was in the integration of both methods in a single framework, but we were specially centred on the CL part. The resulting method, Segment and Learn Semantics (SaLS) works as an efficient and functioning Continual Semantic Segmentation algorithm, and can be integrated in a locomotive robot with the function of a perceptive system that helps the robot to be aware of its vicinity so that it can interact accordingly with it.

In order to achieve this, we had to take some relevant design choices. In terms of the segmentation tool, we decided to use a state-of-the-art algorithm whose objective is to serve as the foundation model of many Computer Vision applications, that is, to be used in different tasks with little to no training. This tool is Segment Anything, that was joint with a powerful tracker, DeAOT, in the open-source project Segment and Track Anything, so that it could segment objects in continuous streams of frames in an efficient and accurate way.

Regarding the classification and learning, we decided to use a parameter isolation method called Progressive Neural Networks, proved to alleviate the catastrophic forgetting from which many CL algorithms suffer. However, this algorithm was aimed to be used in a task-incremental setting, but knowing the task ID when inferring new objects is usually not possible, so we had to convert the PNN in a class-incremental method instead. To fulfill so, we decided to train a new column for every task: each Neural Network that form the PNN is trained with objects belonging to only one category. Consequently, when inferring a new object whose category we ignore, only one of the columns returns a positive result and the object is identified belong to the corresponding category. The other issue to tackle is how to know if an object is unknown. For that reason we employed an Out-of-Distribution detection method, MaxLogits, that uses the unnormalised output logits from the multiple NNs in the PNN as anomaly scores. With these two modifications, we achieved a class-incremental and task-agnostic CL algorithm.

In order to ensure and validate that the decisions taken in the PNN were pertinent, we did some experiments comparing different settings for the input data transformations, backbone and base network of the PNN, and OOD detection method. The results of the first experiment showed that cropping the input images to a resolution around 300

pixels is the optimal for having the best classification performance without overloading the available computational resources. In the second experiment we compared a reliable backbone, the ResNet-50 – commonly used in Computer Vision tasks as a powerful feature extractor and for image classification – and another state-of-the-art tool, DINOv2, also meant to be a foundation model for Computer Vision tasks in the same way as SAM. DINOv2 was used as the backbone of the PNN accompanied by a 7-layer NN functioning as a linear classifier, and it demonstrated the best performance over the different configurations for PNN, reaching the best accuracy while occupying the least space in disk and GPU. In the third experiment we compared two different OOD detection methods, MaxLogit and MSP, but the former had the best results with a 20% of $FPR95$ and over 95% for both $AUROC$ and $AUPR$, having the Binary Cross Entropy loss function as the training criterion.

Afterwards, we evaluated the performance of the PNN in a CL setting using the dataset collected from TORO. The experiment resulted in an average accuracy across all the tasks of 78.86% for the known classes, and 70.78% of the OOD samples were correctly identified as unknown objects. No sign of catastrophic forgetting was shown, with the only exception of a decrease in accuracy of one category after training a second category because of their visual similarity under some circumstances, such as poor lightning or rapid camera movements. This problem accompanied the results in the rest of experiments, and the proposed solution when using the method is to carefully examine the training and validation sets. We took random images in every iteration, so that the results were not biased, but in a real setting it would be beneficial to choose the most representative samples of each category for training and validation, so that the confusions between categories are avoided.

We also wanted to see the performance of the PNN in an extreme case, that is, when training a high number of tasks. We were mainly interested in the computational resources consumed by the algorithm, and according to the results it is feasible to do so. The model increases its size in disk with each column, as the base network is copied for every new task, concretely 0.834 MBytes per task. Training 200 tasks increases the size up to 250 MBytes, which we believe is not too much and almost every mobile robotic system nowadays would be able to store it. However, the GPU memory is also increased with every task, and in this case the slope is of around 2.5 MBytes per task, so this might be the most limiting factor. Since we used random images to train in this experiment because the total number of objects in the TORO dataset is low, it would be interesting to see the accuracy performance in a huge dataset, such as IMAGENET.

Once the CL part is validated, we joined it with SAM-Track to create the full pipeline with segmentation, classification and learning power. This method, Segment and Learn Semantics, has also been evaluated using the seven different sequences collected with the TORO camera. These sequences have been semi-automatically annotated using SaLS, and this is an additional application of the algorithm. The segmentation of the frames is automatic, and the user just needs to input the semantic label of the object. With the help of the tracking, most of the subsequent frames do not need to be manually labelled, so the method proposed in this thesis might be used as a powerful video object annotation tool.

The evaluation of the method started with the processing of the sequences and

continued with the posterior comparison of the classification results with the ground truth. For the known, pre-trained main terrains, the average accuracy score among all the sequences was of 75.54% of objects correctly segmented and classified in the different sequences. The unknown objects, which were learned online, displayed an average accuracy of 84.75%. Hence, the total average accuracy of the objects appearing in the frames of the sequences were between 71.21% and 92.25%. These scores validate completely the classification performance of the algorithm for the majority of frames; not only the previously known objects display a notable score, but also the newly learned objects. Some of the main classification errors that we detected were also related to the similarity between some object frames, or due to the complexity in the shape of unknown objects. It is worth noting that if an object is not correctly segmented, the error will be propagated and the classification – which takes the resulting masked object as input – will probably be wrong too. The confusion matrix resulting in this experiment also demonstrated a remarkable performance, with most of the categories being correctly classified in more of 90% of the cases. Regarding the computational resources, by using the largest models for SAM and DeAOT, only a maximum of 6 GBytes of GPU was used, and the training and inference times were low, around 30 seconds and 0.1 seconds, respectively. These values can be decreased if other segmentation models are used, but with the current configuration SaLS is able to process around 10 fps, so it could be implemented in a real-time system that does not require extremely fast updates.

These results prove that the framework of Continual Semantic Segmentation works with a notable performance. The computational resources required are limited by the GPU availability, which is a fact to take into account for future applications, specially in mobile robotics which may lack of GPU or have a small one. In our experiments, we have not processed the data in the TORO robot, but we did so in a ground station connected to the system and receiving the information from the camera. Ideally, it would be beneficial to implement the method in the processing unit of the robot, so that it can work in an independent manner.

We did an extensive analysis of the CL capability of the proposed method, and we validated its performance in a collected dataset, using real images obtained from a mobile robotic system. However, there are still some lines of research that would be worth to have in mind. Regarding the segmentation tool, SAM-Track is a relatively new method so there is still room for improvements and new implementations. For instance, SAM is quite heavy and its processing time exceeds the expectations for a real-time application. One way of alleviating this would be to use a lighter image encoder, such as FastSAM or MobileSAM, which are projects that have decreased the size of SAM with the handicap of also decreasing their encoding capabilities. It would be interesting to try different segmentation tools, but we have proved with SAM and DINOv2 that these kind of algorithms can work as foundation models in a very efficient way.

One relevant point in the segmentation side is the possibility of using different modes and input prompts. For simplicity and in order to focus in the CL, we have chosen a single point as prompt, but other possibilities include boxes, multiple points, or even text prompts. Combining these with CL might be a fascinating line of research

that we think could be explored in the future.

Regarding the PNN and the CL, there are also some interesting points that could be further discussed and researched. There were different design decisions that had more than one possibility, for instance, the conversion from task- to class-incremental of the PNN, the configuration of backbone and base network, the OOD detection method, or the whole CL algorithm. There are multiple methods for each one on the mentioned aspects that could fit and be efficient in this context. As an example, a CL method belonging to other family, such as the regularization group, could be used instead of PNN. We chose PNN in the first stance because other works validated its performance and it was proved to alleviate catastrophic forgetting, however, other algorithms could work with the same efficient performance. Regarding the OOD method, a more efficient detection could be implemented by doing temperature scaling and calibrating the anomaly scores.

Additionally, we mentioned that sometimes the method detected as unknown some known objects, depending on the part of the object that was segmented or the perspective of the camera. In these cases, we have the possibility to train another new head for the same category, even though it is already supposed to be known, or skip the training and keep the known object as unknown. The implications that both options might have would be an interesting path of research, as they have different implications and might impact the performance of the algorithm in different ways.

As a last note, we would like to address the possibility of integrating SaLS as the vision system in TORO. Due to the lack of time, we could not fully test the method in the robot in real-time. For that purpose, in the method side it would be necessary to add the data from the depth estimation sensor, as with only the semantics TORO would not be able to compute how close or far the terrains are. It would signify a great advancement for a vision system, since it would behave even more similarly to that of a human: the robot would recognize a terrain, decide the walking parameters needed to step over it, calculate the distance between it and itself, and adjust the foot step accordingly.

References

[1] A. Al-Rahayfeh and M. Faezipour, "Enhanced frame rate for real-time eye tracking using circular hough transform," in *2013 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. IEEE, 2013, pp. 1–6.

[2] J. Balaram, M. Aung, and M. P. Golombek, "The ingenuity helicopter on the perseverance rover," *Space Science Reviews*, vol. 217, no. 4, p. 56, 2021.

[3] P. Baldi and P. J. Sadowski, "Understanding dropout," *Advances in neural information processing systems*, vol. 26, 2013.

[4] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "Semantickitti: A dataset for semantic scene understanding of lidar sequences," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9297–9307.

[5] G. Bhat, F. J. Lawin, M. Danelljan, A. Robinson, M. Felsberg, L. Van Gool, and R. Timofte, "Learning what to learn for video object segmentation," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 777–794.

[6] W. Boerdijk, M. Durner, M. Sundermeyer, and R. Triebel, "Towards robust perception of unknown objects in the wild," in *ICRA 2022 workshop on "Robotic Perception and Mapping: Emerging Techniques"*, 2022.

[7] W. Boerdijk, M. G. Müller, M. Durner, M. Sundermeyer, W. Friedl, A. Gawel, W. Stürzl, Z.-C. Marton, R. Siegwart, and R. Triebel, "Rock instance segmentation from synthetic images for planetary exploration missions," in *2021 IEE/RSJ International Conference on Intelligent Robots and Systems, IROS (Workshops)*, 2021.

[8] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[9] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[10] F. Cermelli, M. Cord, and A. Douillard, "Comformer: Continual learning in semantic and panoptic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3010–3020.

[11] Y. Cheng, L. Li, Y. Xu, X. Li, Z. Yang, W. Wang, and Y. Yang, "Segment and track anything," *arXiv preprint arXiv:2305.06558*, 2023.

[12] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[13] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3366–3385, 2021.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[15] M. Denninger and R. Triebel, "Persistent anytime learning of objects from unseen classes," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4075–4082.

[16] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni, "Don't forget, there is more than forgetting: new metrics for continual learning," *arXiv preprint arXiv:1810.13166*, 2018.

[17] A. Douillard, Y. Chen, A. Dapogny, and M. Cord, "Plop: Learning without forgetting for continual semantic segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 4040–4050.

[18] M. Durner, W. Boerdijk, M. Sundermeyer, W. Friedl, Z.-C. Márton, and R. Triebel, "Unknown object segmentation from stereo images," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 4823–4830.

[19] J. Englsberger, A. Werner, C. Ott, B. Henze, M. A. Roa, G. Garofalo, R. Burger, A. Beyer, O. Eiberger, K. Schmid *et al.*, "Overview of the torque-controlled humanoid robot toro," in *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2014, pp. 916–923.

[20] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, pp. 98–136, 2015.

[21] K. A. Farley, K. H. Williford, K. M. Stack, R. Bhartia, A. Chen, M. de la Torre, K. Hand, Y. Goreva, C. D. Herd, R. Hueso *et al.*, "Mars 2020 mission overview," *Space Science Reviews*, vol. 216, pp. 1–41, 2020.

[22] Y. Federigi, "Balancing and static walking control for a compliantly actuated quadruped exploiting system inherent elasticities," Ph.D. dissertation, University of Pisa, 2017.

[23] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[25] D. Hendrycks, S. Basart, M. Mazeika, A. Zou, J. Kwon, M. Mostajabi, J. Steinhardt, and D. Song, "Scaling out-of-distribution detection for real-world settings," *arXiv preprint arXiv:1911.11132*, 2019.

[26] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," *arXiv preprint arXiv:1610.02136*, 2016.

[27] B. Henze, A. Dietrich, M. A. Roa, and C. Ott, "Multi-contact balancing of humanoid robots in confined spaces: Utilizing knee contacts," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 697–704.

[28] B. Henze, M. A. Roa, and C. Ott, "Passivity-based whole-body balancing for torque-controlled humanoid robots in multi-contact scenarios," *The International Journal of Robotics Research*, vol. 35, no. 12, pp. 1522–1543, 2016.

[29] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, "Hands-on bayesian neural networks—a tutorial for deep learning users," *IEEE Computational Intelligence Magazine*, vol. 17, no. 2, pp. 29–48, 2022.

[30] T. Kalb, M. Roschani, M. Ruf, and J. Beyerer, "Continual learning for class-and domain-incremental semantic segmentation," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 1345–1351.

[31] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[32] C. Li, J. Liu, X. Ren, W. Zuo, X. Tan, W. Wen, H. Li, L. Mu, Y. Su, H. Zhang *et al.*, "The chang'e 3 mission overview," *Space Science Reviews*, vol. 190, pp. 85–101, 2015.

[33] C. Li, W. Zuo, W. Wen, X. Zeng, X. Gao, Y. Liu, Q. Fu, Z. Zhang, Y. Su, X. Ren *et al.*, "Overview of the chang'e-4 mission: Opening the frontier of scientific exploration of the lunar far side," *Space Science Reviews*, vol. 217, pp. 1–32, 2021.

[34] Y. Liao, J. Xie, and A. Geiger, "Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3292–3310, 2022.

[35] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, C. Li, J. Yang, H. Su, J. Zhu *et al.*, "Grounding dino: Marrying dino with grounded pre-training for open-set object detection," *arXiv preprint arXiv:2303.05499*, 2023.

[36] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner, "Online continual learning in image classification: An empirical survey," *Neurocomputing*, vol. 469, pp. 28–51, 2022.

[37] A. Maracani, U. Michieli, M. Toldo, and P. Zanuttigh, "Recall: Replay-based continual learning in semantic segmentation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 7026–7035.

[38] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3523–3542, 2021.

[39] A. Nazemi, Z. Moustafa, and P. Fieguth, "Clvos23: A long video object segmentation dataset for continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2495–2504.

[40] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 3400–3407.

[41] M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, R. Howes, P.-Y. Huang, H. Xu, V. Sharma, S.-W. Li, W. Galuba, M. Rabbat, M. Assran, N. Ballas, G. Synnaeve, I. Misra, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "Dinov2: Learning robust visual features without supervision," 2023.

[42] R.-G. Paolo, A.-S. Alin, and G. Hirzinger, "On the kinematic modeling and control of a mobile platform equipped with steering wheels and movable legs," in *Proceedings*, 2009.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung, "A benchmark dataset and evaluation methodology for video object segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 724–732.

[46] M. M. Petrou and C. Petrou, *Image processing: the fundamentals*. John Wiley & Sons, 2010.

[47] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[48] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on robot learning*. PMLR, 2017, pp. 262–270.

[49] D. Schnaus, "Progressive bayesian neural networks," Master's thesis, Technical University of Munich, 2021.

[50] M. J. Schuster, M. G. Müller, S. G. Brunner, H. Lehner, P. Lehner, R. Sakagami, A. Dömel, L. Meyer, B. Vodermayer, R. Giubilato *et al.*, "The arches space-analogue demonstration mission: Towards heterogeneous teams of autonomous robots for collaborative scientific sampling in planetary exploration," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5315–5322, 2020.

[51] Q. She, F. Feng, X. Hao, Q. Yang, C. Lan, V. Lomonaco, X. Shi, Z. Wang, Y. Guo, Y. Zhang *et al.*, "Openloris-object: A robotic vision dataset and benchmark for lifelong deep learning," in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 4767–4773.

[52] Q. She, F. Feng, Q. Liu, R. H. Chan, X. Hao, C. Lan, Q. Yang, V. Lomonaco, G. I. Parisi, H. Bae *et al.*, "Iros 2019 lifelong robotic vision challenge–lifelong object recognition report," *arXiv preprint arXiv:2004.14774*, 2020.

[53] V. Subramanian, *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. Packt Publishing, 2018.

[54] A. M. Sundaram, B. Henze, O. Porges, Z.-C. Márton, and M. A. Roa, "Autonomous bipedal humanoid grasping with base repositioning and whole-body control," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 395–402.

[55] A. R. Vasavada, "Mission overview and scientific contributions from the mars science laboratory curiosity rover after eight years of surface operations," *Space Science Reviews*, vol. 218, no. 3, p. 14, 2022.

[56] N. Vödisch, K. Petek, W. Burgard, and A. Valada, "Codeps: Online continual learning for depth estimation and panoptic segmentation," *arXiv preprint arXiv:2303.10147*, 2023.

[57] T. S. Wang, Z.-C. Marton, M. Brucker, and R. Triebel, "How robots learn to classify new objects trained from small data sets," in *Conference on Robot Learning*. PMLR, 2017, pp. 408–417.

[58] A. Wedler, M. G. Müller, M. Schuster, M. Durner, S. Brunner, P. Lehner, H. Lehner, A. Dömel, M. Vayugundla, F. Steidle *et al.*, "Preliminary results for

the multi-robot, multi-partner, multi-mission, planetary exploration analogue campaign on mount etna," in *Proceedings of the International Astronautical Congress, IAC*, 2021.

[59] A. Wedler, M. G. Müller, M. Schuster, M. Durner, P. Lehner, A. Dömel, F. Steidle, M. Vayugundla, R. Sakagami, L. Meyer *et al.*, "Finally! insights into the arches lunar planetary exploration analogue campaign on etna in summer 2022," in *73rd International Astronautical Congress, IAC 2022*, 2022.

[60] N. Xu, L. Yang, Y. Fan, D. Yue, Y. Liang, J. Yang, and T. Huang, "Youtube-vos: A large-scale video object segmentation benchmark," *arXiv preprint arXiv:1809.03327*, 2018.

[61] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," *arXiv preprint arXiv:2110.11334*, 2021.

[62] Z. Yang, Y. Wei, and Y. Yang, "Associating objects with transformers for video object segmentation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[63] Z. Yang and Y. Yang, "Decoupling features in hierarchical propagation for video object segmentation," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[64] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2636–2645.

[65] H.-y. Zhao, Q.-y. Ma, J. Cao, and Q.-k. Chen, "Dynamic neural network for incremental learning with task extended: Research progress and prospect," *ACTA ELECTONICA SINICA*, p. 1, 2023.

[66] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 633–641.

A TORO dataset images

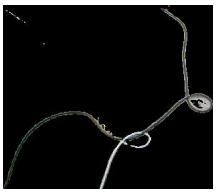
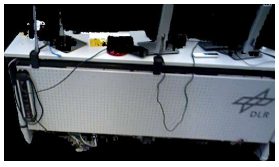
Category	Image	Category	Image
dark_mat		plane_wall	
lab_floor		extinguisher	
stones		cable	
ramp		TV_mount	
blue_mattress		bistro_table	
pedestal		person	
wall		stand_board	
PC_table			

Table A1: Examples of each category of the dataset collected with TORO camera.

B Continual Learning accuracy matrices

Apart from the accuracy results in the CL part of the method, shown in Figure 20, we expose in the following sections the accuracy matrices for both the known categories and for the OOD samples.

B.1 Class accuracy matrix

This Table B1 presents the accuracy values of the test set of task i after training from task 1 to task j. Note that the elements above the diagonal are null, as there is no backward knowledge transfer. A degradation on the accuracy of the test set of task 0 is observed after training task 1, and after task 3.

a	T_{e_0}	T_{e_1}	T_{e_2}	T_{e_3}	T_{e_4}	T_{e_5}
T_{r_0}	97.67	0	0	0	0	0
T_{r_1}	72.67	67.67	0	0	0	0
T_{r_2}	72.67	67.67	91.89	0	0	0
T_{r_3}	71.33	67.67	91.89	86.74	0	0
T_{r_4}	71.00	67.67	91.89	86.74	76.85	0
T_{r_5}	71.00	67.67	91.89	86.74	76.85	80.00

Table B1

B.2 OOD accuracy matrix

The Table B2 displays the accuracy on the OOD samples after training with certain task i and testing in the test sets of the subsequent tasks j. The diagonal and the elements below it are null, since the tasks have already been trained and theit elements become known for the PNN.

a_{OOD}	T_{e_0}	T_{e_1}	T_{e_2}	T_{e_3}	T_{e_4}	T_{e_5}
T_{r_0}	0	31.00	71.85	42.85	60.56	100.00
T_{r_1}	0	0	68.13	40.62	61.33	100.00
T_{r_2}	0	0	0	40.62	61.33	100.00
T_{r_3}	0	0	0	0	83.43	100.00
T_{r_4}	0	0	0	0	0	100.00
T_{r_5}	0	0	0	0	0	0

Table B2

C Segment and Learn Semantics pseudo-code

C.1 SaLS pseudo-code

As an extension of the SaLS method diagram in Figure 14, we display here the pseudo-code of the SaLS algorithm.

Algorithm 1 Segment and Learn Semantics

```
for Every frame with index frame_idx do
    if Automatic mode then
        if frame_idx % sam_gap = 0 then
            Run automatic segmentation
            Track objects
            Find new objects and update tracker
        else
            Track objects
        end if
    else if Interactive mode then
        if Area around point prompt is not segmented then
            Restart tracker
            Run point prompt segmentation
            Add object to tracker
        else
            Track object
        end if
    end if
    Crop and save objects in mask
    Run PNN to classify objects
    if Unknown object/s then
        Ask user for label/s
        Train PNN with new object/s
        Re-classify object/s
    end if
    Display or save segmented, classified frame
end for
```

D Results on consecutive frames

Expanding the example results that have been shown in Section 4.2.2, we present in this Annex some other results obtained after processing the different sequences with SaLS. We want to display consecutive frames from the same sequence, to illustrate how the algorithm is able to detect different objects, in Figures D1, D2, D3 and D4.

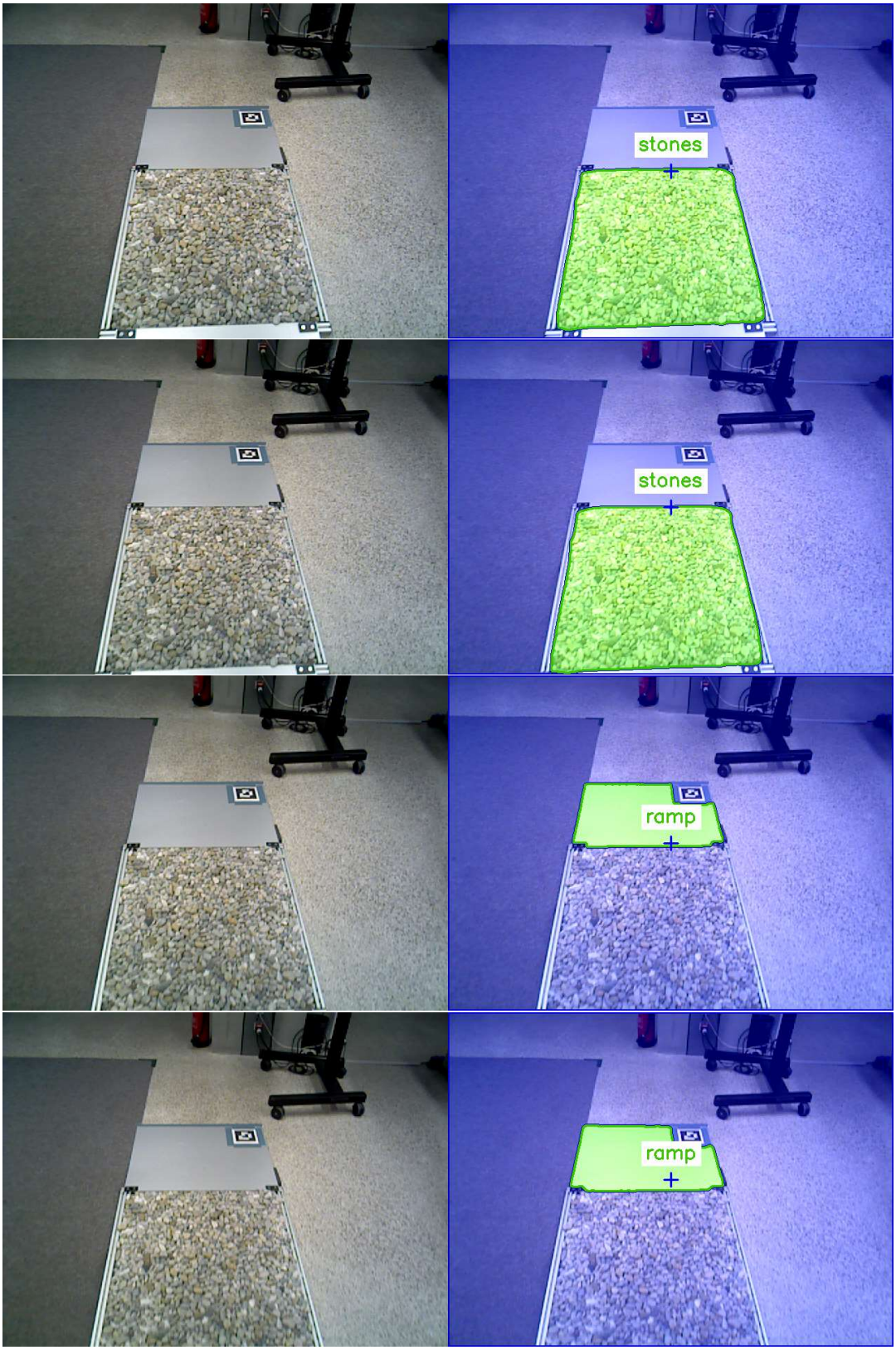


Figure D1: Frames in sequence 2: the algorithm detects the change from "stones" to "ramp", and classifies the terrains correctly.

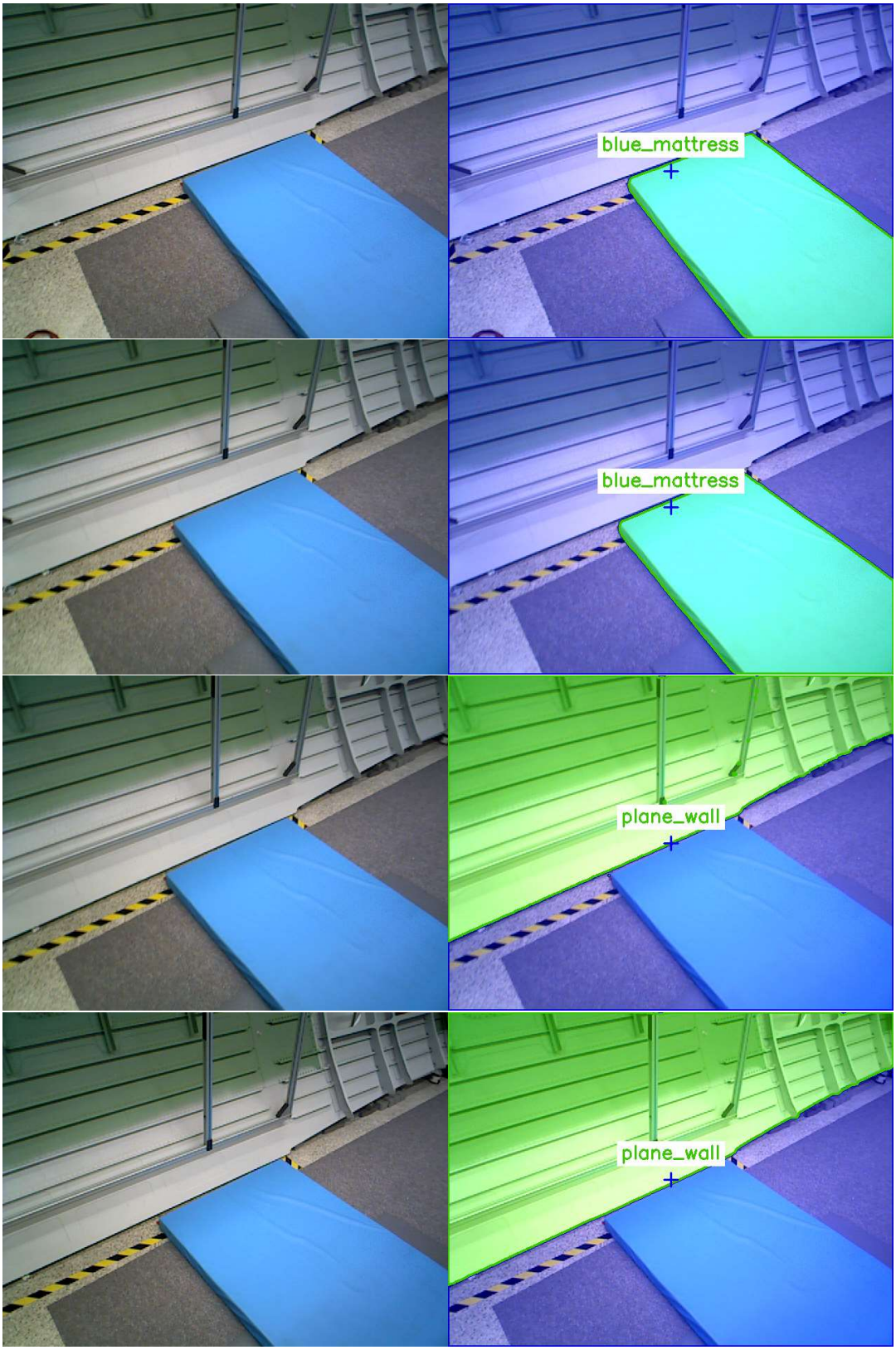


Figure D2: Frames in sequence 3: the algorithm detects the change from "blue_mattress" to "plane_wall", and classifies the objects correctly.

Figure D3: Frames in sequence 6: the algorithm detects the change from "pedestal" to "dark_mat", and classifies the terrains correctly.

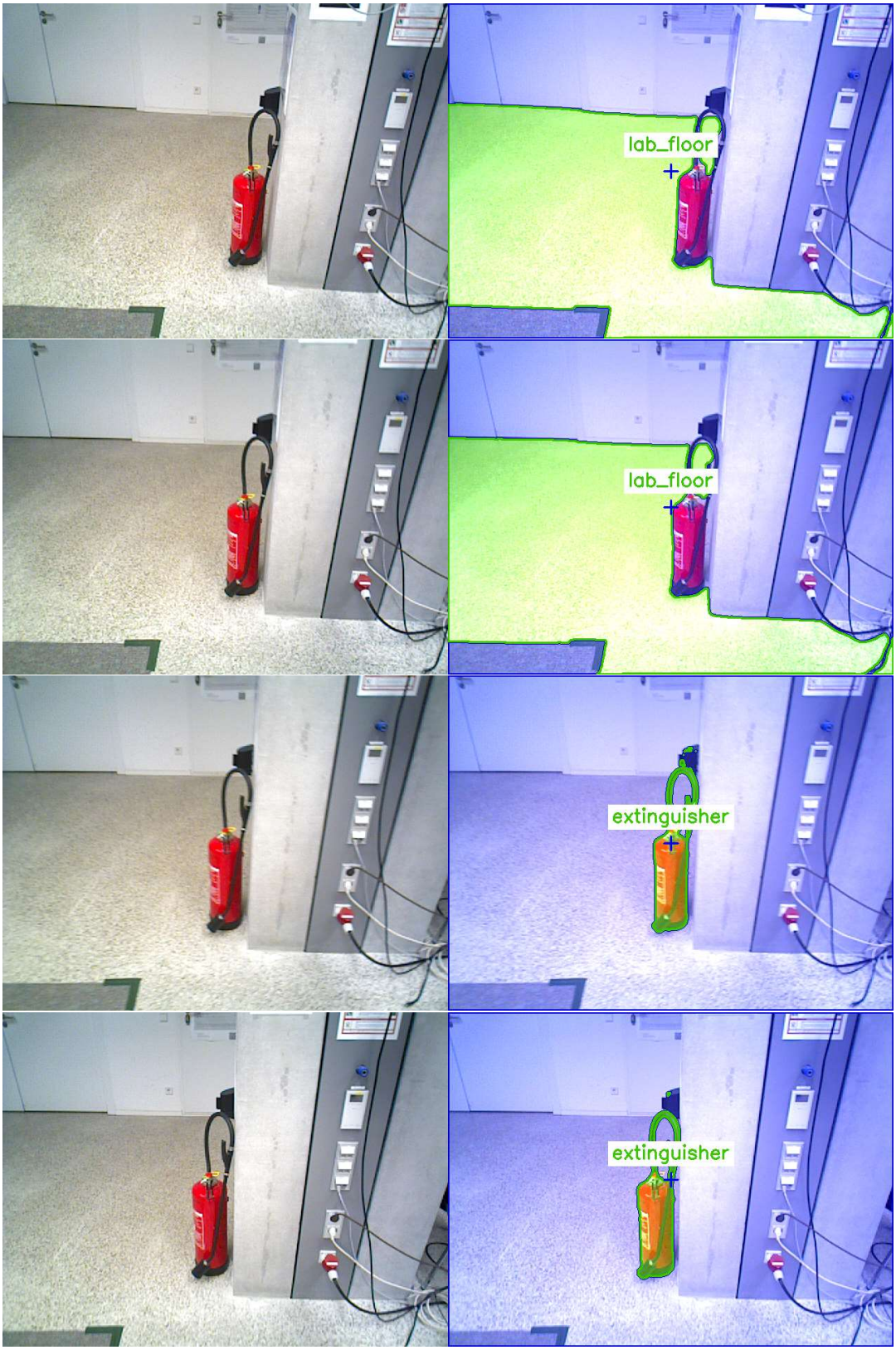


Figure D4: Frames in sequence 7: the algorithm detects the change from "lab_floor" to "extinguisher", and classifies the objects correctly.