Semi-supervised Learning From Demonstration through Program Synthesis: An Inspection Robot Case Study

Semi-supervised Learning From Demonstration Through Program Synthesis: An Inspection Robot Case Study

Simón C. Smith
Institute of Perception, Action and Behaviour
School of Informatics
The University of Edinburgh*
Edinburgh, United Kingdom
artificialsimon@ed.ac.uk

Subramanian Ramamoorthy
Institute of Perception, Action and Behaviour
School of Informatics
The University of Edinburgh
Edinburgh, United Kingdom
s.ramamoorthy@ed.ac.uk

Semi-supervised learning improves the performance of supervised machine learning by leveraging methods from unsupervised learning to extract information not explicitly available in the labels. Through the design of a system that enables a robot to learn inspection strategies from a human operator, we present a hybrid semi-supervised system capable of learning interpretable and verifiable models from demonstrations. The system induces a controller program by learning from immersive demonstrations using sequential importance sampling. These visual servo controllers are parametrised by proportional gains and are visually verifiable through observation of the robot's position in the environment. Clustering and effective particle size filtering allows the system to discover goals in the state space. These goals are used to label the original demonstration for end-to-end learning of behavioural models. The behavioural models are used for autonomous model predictive control and scrutinised for explanations. We implement causal sensitivity analysis to identify salient objects and generate counterfactual conditional explanations. These features enable decision making interpretation and post hoc discovery of the causes of a failure. The proposed system expands on previous approaches to program synthesis by incorporating repellers in the attribution prior of the sampling process. We successfully learn the hybrid system from an inspection scenario where an unmanned ground vehicle has to inspect, in a specific order, different areas of the environment. The system induces an interpretable computer program of the demonstration that can be synthesised to produce novel inspection behaviours. Importantly, the robot successfully runs the synthesised program on an unseen configuration of the environment while presenting explanations of its autonomous behaviour.

1 Introduction

In recent years, we have seen robots cross the chasms between laboratory testbeds and field deployments, increasingly involving operation alongside human co-workers. A particularly useful domain of application is that of robots that take over human tasks in hazardous environments such as offshore energy platforms [28]. In contrast to more simple environments, such hazardous environment have a set of particular requirements for a robotic system to be successfully implemented. For example, to quickly respond to emergencies or to keep a vital part of the system running when a change in the environment has occurred. One must be able to rapidly program these robots to solve a range of tasks. One approach to rapidly reconfiguring robots for new tasks is to enable a human operator to teach the robot, by demonstrating the task and performing it rather than by explicitly writing programs. This is the approach of learning from demonstration (LfD). These task reconfigurations can be at the level of low-level motor commands or they may be at a higher level of abstraction as a combination of low-level tasks. When

*This work was supported by funding from the ORCA Hub EPSRC project (EP/R026173/1, 2017-2021).

R.C. Cardoso, A. Ferrando, D. Briola, C. Menghi,
T. Ahlbrecht (Eds.): First Workshop on Agents and
Robots for reliable Engineered Autonomy 2020 (AREA'20).
EPTCS 319, 2020, pp. 81–101, doi:10.4204/EPTCS.319.7

enough demonstration data is available, low-level tasks can be learned through non-linear regression techniques, like deep learning [8,22]. These learning techniques present consistently higher performance when compared to other learning approaches [25,61]. A downside is that such techniques have promoted performance advances at the expense of interpretability and flexibility [21,58]. Robotic systems trained with deep learning based LfD may be inscrutable to the typical operator, hence defeating the point of easy reconfigurability. In this sense, interpretability, flexibility and explainability are desiderata for the class of autonomous systems that will be deployed in such complex scenarios alongside humans.

To improve flexibility, compositional techniques can be used to divide demonstrations into sub-tasks. Learning composition of individual motions usually requires a set of primitives that are sequenced to solve more complex tasks [18,44]. Usually, these approaches require the primitives to be known before learning [7]. Approaches that automatically segment a demonstration [26,34] still have to deal with an unknown number of segments and task-dependent primitives.

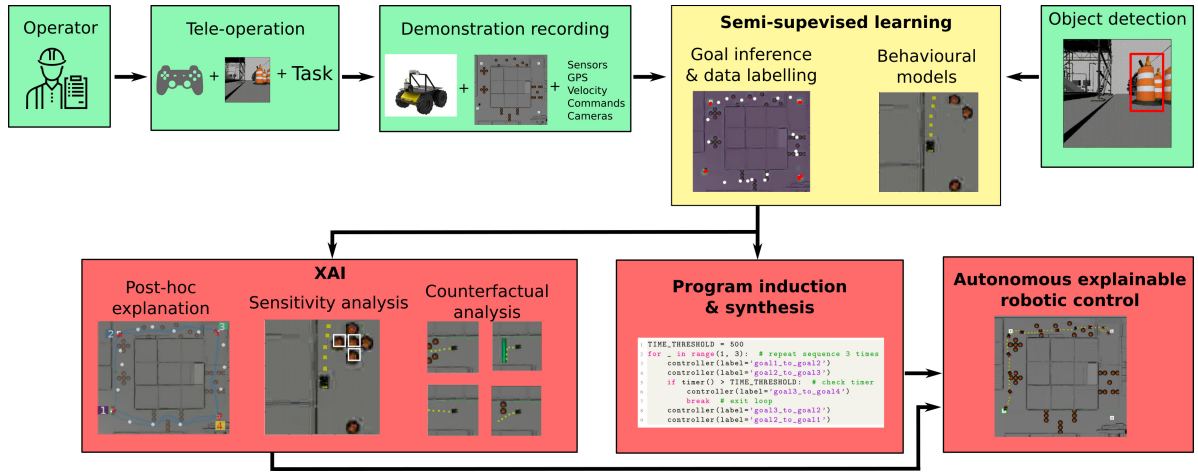
We present a semi-supervised learning system that can induce a program from a demonstration, synthesise new programs and explain its behaviour. The unsupervised part of the system discovers proportional controllers that satisfy the demonstration. Our system can discover behavioural goals in the state space. The goals, visiting order and proportional controllers allow us to induce a computer program that abstracts the demonstration. This abstraction is a complexity reduction of the demonstration as it allows a user with programming skill to examine the full demonstration at once [40].

Continuing with the semi-supervised paradigm, we use the goals to label and segment the original demonstration. The newly labelled data is used to train end-to-end low-level behavioural model predictors. These behavioural models have two main objectives. First, following a hybrid system of high- and low-level abstractions [36], the behavioural predictions can be used to synthesise new programs that are verifiable by the robot even in unseen configurations of the environment. Second, using black-box analysis [58,62], the behavioural models can explain the actual behaviour based on saliency in the input state of the end-to-end model and by counterfactual explanations.

As mentioned before, this work builds on an existing probabilistic goal identification and program induction approach [11]. However, this approach is particularly limited by the assumption that objects in a given scene are attractors to a robot. The likelihood of a position in an environment being a goal or target is assumed to be proportional to the saliency of the objects or regions in an image. The present work addresses this limitation by introducing repellers that allow the system to account for obstacles in the path of the robot. We extend the system by adding a likelihood term that includes the probability that a particle belongs to the path over the next steps of the robot. Results show that using this approach, our system can learn object avoidance behaviours without explicitly defining these as a high-level goal. As another extension to the original method, we are able to autonomously control the robot to follow a synthesised program containing behaviours unseen in the demonstrations. We test our system in an oil rig digital twin [48], where an operator demonstrates an inspection task in an immersive teleoperation scenario. To summarise, our system can:

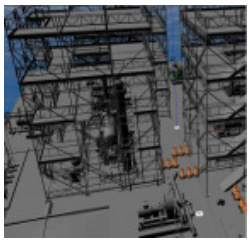
- Automatically infer high-level goals in a surveillance task demonstrated by an operator
- Induce a computer program based on the demonstrations
- Automatic demonstration data labelling using unsupervised learning
- Learn behavioural models for predictive control (MPC) and explainability from the automatic labelled data
- Behaviour prediction, highlighting of prediction sensitive objects and counterfactual queries as post hoc explanations

Fig. 1(a) shows a diagram of the architecture of the system with all the modules and their interactions.

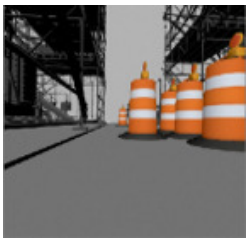


(a) System architecture

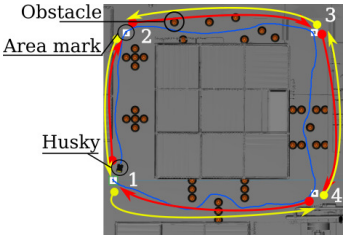
(b) Husky robot



(c) Digital twin



(d) On-board camera



(e) Demonstration

Figure 1: Image (a) shows the architecture of the system. The green blocks represent activities to record the operator demonstrations and the object detection model that are inputs to the inference system. The yellow block represents the module of the system that trains the models, automatically labels the demonstration data and performs goal inference. The red blocks are outputs of the system including the explainable AI module (XAI), program induction and synthesis, and finally autonomous robot control. Image (b) shows the *Husky* robot in the Gazebo simulator. (c) represents the oil rig digital twin [48]. Image (d) shows the on-board camera used by the operator as a visual signal. (e) is the top-view of the oil rig, the operator is asked to manoeuvre the robot from one area to the next following the red arrows, starting on area 1. When the robot reaches area 1 again it reverts the sequence (yellow U-turn arrow). The blue line is the actual path that the operator followed for one demonstration (showing only the first half in the image).

2 Related Work

Learning from demonstration (LfD) is a useful paradigm for robot programming [3]. Initial work in this area focused on direct replication of motions. Since then, the focus has shifted to more general schemes aimed at producing more robust policies [4]. These approaches include linear dynamical attractor systems [20], dynamic motion primitives [49], conditionally linear Gaussian models [15, 37] and sparse online Gaussian processes [12, 27].

Trajectory optimisation approaches have been extended to end-to-end learning with deep neural networks [38], resulting in robust task level visuomotor control through guided policy search. End-to-end learning has facilitated one-shot learning for domain transfer from human video demonstration [70] and

the use of reinforcement learning for optimised control policies [53, 73].

End-to-end learning shows consistent improvement performance in several areas, but some of its drawbacks include a lack of interpretability, the significant amount of training data required [51], and the fact that such models may not easily exhibit features we associate with higher-level conceptual learning and reasoning [32, 35].

Another obstacle in end-to-end systems is a lack of flexibility. This drawback is especially noticeable when the goal of a task is modified [36], or when a systematic difference between test and training data is present [45]. Common means of addressing these challenges are hybrid systems combining high-level symbolic reasoning with sub-symbolic machine learning systems [11, 36]. In this category of combined systems, semi-supervised learning is used to extract high-level structures in an unsupervised way and then use the extracted information to learn low-level representations in a supervised fashion [14].

Semi-supervised learning approaches include self-training methods [6], generative models [1, 64], and graph- and vector-based methods [5]. The most common objective in semi-supervised learning is to directly improve the performance of the supervised learning part [24, 39]. The information that can be extracted in the unsupervised training process is assessed by the increase of the performance in the supervised learning counterpart [72]. In this paper, we use semi-supervised learning to synthesise controllers that generalise to new scenarios (Section 4), and for interpretability and explainability (Section 5).

Definitions for explainability and interpretability still remain as an open question in the context of machine learning [40]. What an explanation is, or when a system is more interpretable than another depends on aspects like the complexity of the explanation itself, the capacity of the user to understand the explanation and the role of the assessing user in the cycle of the system. The method that we propose includes several levels of explainability and interpretability based on complexity reduction, post hoc interpretation and decomposability. One way to achieve complexity reduction is to translate full traces of model predictions so it can be examined by a human in a single pass [50, 69, 74]. In our method, we use this type of complexity reduction when we translate a demonstration (seen as a sequence of actions and sensory input) into a functional computer program. Most post hoc interpretations give explanations of predictions made by models in a black-box setting. Example of this type of explanation are sensitivity analysis in image classification [56, 63, 68]. We combine object recognition and sensitivity analysis to highlight the objects that are taken into consideration by the MPC. Decomposability explains the parts of the system (input data, calculation and parameters) in separated ways making them more intelligibles [42, 59]. In the case of our system, the program induction and MPC are based on the decomposition of the demonstration as visually grounded goals.

3 Goal Inference and Program Induction

The first objective of our system is to identify high-level goals within a demonstration. In general, goals can be defined in different spaces of the system either explicitly or implicitly, and are task-dependent. For example, they could be key positions in an environment, the pose or joint configuration of a robot arm, continuous application of force, or some unknown reward function [47]. In this work, we consider a task requiring a robot to inspect an industrial oil rig platform. The robot has to visit pre-defined areas in the environment in a certain order. For a single demonstration, a user teleoperates a robot to visit these areas in the required order. The system then searches for goals within this demonstration trace, identifying visually salient dynamical attractors (or repellers) in the environment. First, using a probabilistic generative model, the system infers low-level controllers. Then, these controllers are visually grounded using a perception network to allow generalisation among different demonstrations

and to distinguish between transient and attractor goals.

The scenario where we test our approach is an immersive teleoperation inspection task in a digital oil rig twin [48]. In this scenario, an operator drives an unmanned ground vehicle [57] for inspection of different areas of the platform. The operator inspects four areas in a cyclic order, reversing direction when it reaches the starting area (Fig. 1(e)). A demonstration trace includes the position of the robot in the scene, the actions are chosen by the operator (move forward, backwards, rotate to the left, rotate to the right), and images from top-view and on-board cameras recorded at 10Hz.

3.1 Sequential Importance Sampling with Attribution Prior

To find the goals, we model the demonstration as a switching high-level task comprised of sub-tasks implemented by low-level controllers. Following [11], we assume that any task can be modelled as a set of proportional controllers. The state-space includes the horizontal and vertical dimensions on a top-view image of the full scenario (Fig. 1(e)), and the rotation range in the horizontal plane relative to the robot. The proportional controllers determine the linear and angular velocities of the robot:

$$\mathbf{u}_{\mathbf{x}} = K_p^j \left(\mathbf{x} - \mathbf{x}_d^j \right), \tag{1}$$

with $j = 1 \ldots J$ denoting a sequence of controllers, $\mathbf{x}$ as the actual state, and $\mathbf{x}_d^j$ as the objective state and gains K_p^j of controller j.

As part of the controller inference, we model the influence of the objects in the demonstrations. The operator can guide the robot towards an object that marks the middle of the inspected area, and also avoid objects when traversing from one area to the next. We ground objects with a visual sensory network that takes the top-view image of the environment and outputs the predicted position of the robot depending on the action chosen by the operator. We train a deep convolutional network with data collected from several demonstrations. The training pairs include the image of the environment as input and the following 5 relative positions of the robot in intervals of 2s as the output. After training, we apply causal saliency detection to the image. Saliency detection allows the system to identify what objects are taken into account by the network to predict future positions of the robot and separates them from other non-important objects like the background. We use gradient-based sensitivity map to create a filter that highlights the salient input in the deep neural network position prediction task [62, 71]. We use this filter as the attribution prior Φ for sequential importance sampling. In the prior, we include the Euclidean distance between $\mathbf{x}_d^k$ and the closest position to the trace in the horizontal plane. We include this distance to reduce the probability that the system infers an obstacle as an actual goal.

We model the switch of the actual controller j to the next one $j+1$ as a Bernoulli trial with switch probability p [11]. When a switch occurs, we sample goals and gains from the prior distribution Φ. If no switch has occurred, we sample from Gaussian jitter. The generative model is:

$$\begin{aligned}
k &\sim \text{Bernoulli}(p) \\
\mathbf{x}_d^j(t) &\sim \begin{cases} \mathcal{N}(\mathbf{x}_d^j(t-1), \mathbf{Q}_{\mathbf{x}}) & \text{if } k = 0 \\ \Phi(\mathbf{x}) & \text{if } k = 1 \end{cases} \\
K_p^j(t) &\sim \mathcal{N}(K_p^j(t-1), \mathbf{Q}_{kp}) \\
\mathbf{u}_{\mathbf{x}} &\sim \mathcal{N}(K_p^j(\mathbf{x} - \mathbf{x}_d^j), \mathbf{R}),
\end{aligned}$$

where for the sampling process, we define $\mathbf{Q}_{\mathbf{x}}$ and $\mathbf{Q}_{kp}$ as transition uncertainty terms and $\mathbf{R}$ as the controller noise.

Algorithm 1 describes the sequential importance sampling re-sampling procedure used to infer the controller gains for a trace of T states. Parameter p models the probability of the robot switching to a new controller. For each step of the demonstration, the algorithm samples N particles and evaluates the likelihood of the controller to satisfy the objective $\mathbf{x}_d^j$ from the actual state $\mathbf{x}$. This evaluation is made in two steps. First, for pN particles, where p represents the probability that the algorithm continues with the same controller to complete the task. The new particle is sampled from the previous time step particle adding Gaussian jitter. Second, for $(1-p)N$ particles, the system samples new particles from the attribution prior Φ.

Algorithm 1 Sequential importance sampling with attribution prior

```
Initialise N particles
for t = 0 to T do
  for k = 0 to pN do
    Sample x_d^k(t) ~ N(x_d^k(t-1), Q_x)
    Sample K_p^k(t) ~ N(K_p^k(t-1), Q_kp)
    Evaluate L^k = N(K_p^j(x - x_d^j), R)
  end for
  for k = pN to N do
    Sample x_d^k(t) ~ Φ(x)
    Sample K_p^k(t) ~ N(K_p^k(t-1), Q_kp)
    Evaluate L^k = N(K_p^j(x - x_d^j), R)
  end for
  Draw N samples: x_d^k, K_p^k ~ L^k
end for
```

As mentioned before, Φ is used as prior in the re-sampling process. An additional benefit of the use of the prior is to reduce the sampling space. We define the likelihood L_a^k as the probability of a particle to be a salient point in a deep neural network model trained to predict the future positions of the robot from an overhead camera. We train the neural network as an end-to-end model $I(\mathbf{x}(t))$, with convolutional and fully-connected layers. We extend the sampling process to re-sample based on an additional attribution likelihood. This new likelihood L_b^k is based on the minimum inverse distance of the particle to the path of the robot, so as to favour particles that are closer to the path taken by the robot. Finally, the attribution prior likelihoods are:

$$L_a^k(\mathbf{x}_e^k) = \frac{\partial I(\mathbf{x}(t))}{\partial I(\mathbf{x}_e^k)}, \tag{2}$$

$$L_b^k(\mathbf{x}_d^k) = \min_{t \le q \le T} \| \mathbf{x}(q) - \mathbf{x}_d^k \|^{-1}, \tag{3}$$

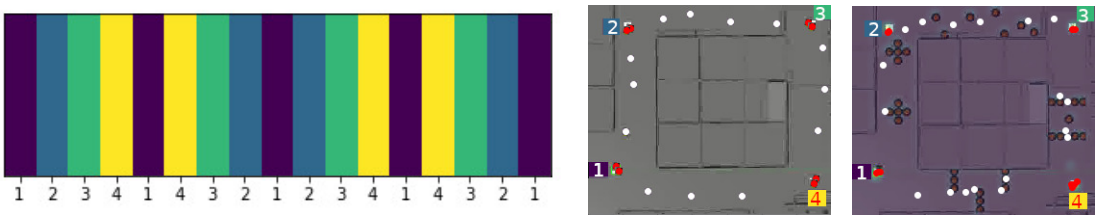
where $\mathbf{x}_e^k$ is the position of the particle in the input image, and $\mathbf{x}_d^k$ is the position of the particle in the scene. Note that for the calculation of L_d^k we only use the horizontal and vertical dimensions of the vector, and do not consider the rotation of the robot. Algorithm 2 shows the re-sampling rules.

After inference over all the steps T, the result is a distribution over possible controllers for each time step. Using the effective particle size [31],

$$N_{\text{eff}} = \frac{1}{\sum_{k=0}^{N} (L^k)^2} \tag{4}$$

Algorithm 2 Extended re-sampling from attribution prior

Draw $k = 1 \ldots N_p$ samples: $\mathbf{x}_d^k \sim [\mathbf{x}(t), \ldots, \mathbf{x}(t+M)]$
Evaluate attribution likelihood: $L_a^k(\mathbf{x}_d^k)$
Draw $k = 1 \ldots N_p$ samples: $\mathbf{x}_d^k \sim L_a^k$
Evaluate attribution likelihood: $L_b^k(\mathbf{x}_d^k)$
Draw $k = 1 \ldots N_p$ samples: $\mathbf{x}_d^k \sim L_b^k$



(a) Controllers (b) No obstacles (c) With obstacles

Figure 2: The colour blocks on (a) represent the sequence of inferred controllers. Each controller is associated with a cluster of visiting goals (red dots) by hand-coding each cluster with numbered colour boxes. The white dots represent transit goals. The extracted sequence of goals labelled by area is $[1,2,3,4,1,4,3,2,1]$ (repeated once) that correspond to the original demonstration. Image (b) is the scenario of a demonstration without obstacles, and (c) with them. The extracted sequence of goals is the same for visiting goals for both demonstrations. We set the constants values for the sampling process at $\mathbf{Q_x} = \mathbf{Q}_{kp} = \mathbf{R} = 0.01$ and $p = 0.5$.

we can isolate a sequence of controllers that constitute the demonstration. The idea behind effective particle size is that, on one hand, when the demonstration is switching to a new goal, the majority of particles will have a low probability mass. The low probability of the particles is represented by low effective particle size. On the other hand, when only a single controller accumulates most of the probability mass, the effective particle size will be greatest, and will have established a clear controller goal and gains. Using a peak detector over the effective particle sizes for all the time steps of the demonstration, we can identify the switching of controllers with their associated goal. We identify a sequence of controllers by maximum a-posteriori controller selection at each of the peaks. Using a K-means algorithm with elbow criterion for the number of clusters, the controllers are grouped to produce a symbolic behavioural trace of the demonstration. We asked the operator to run two full-cycles of the demonstration starting in area 1, following a clock-wise visit of the other corners, and inverting the direction once reached area 1 again (Fig. 1(e)). Figure 2 shows as circles the maximum a-posteriori controllers that the system identified. White circles represent low-saliency goals. We term these goals transit goals. The red circle identify high-saliency points clustered as visiting goals. The visiting goals properly delimit the primitives that define the demonstration and the visiting order.

Once the goals are identified, we induce a program that represents an abstraction of the demonstration (Listing 1). To simplify the program, we search for repetition of sub-sequences (loops) and palindromes. A robot operator (with programming skills) can use this program to examine the demonstration.

```
def program():
  execute(1)
  for j in range(2):  # Demonstration loop
    controller_list = [2,3,4,1]
    count = 0
    for k in range(len(controller_list)*2-1):  # Palindromic sequence
      execute(controller_list[count])
      if k >= len(controller_list)-1:
        count = count-1
      else:
        count = count+1
    execute(1)
  return
```

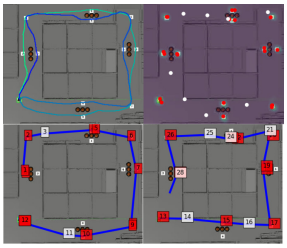
Listing 1: Induced program from demonstration. The *execute(g)* command is a call to a controller that drives the robot to the g goal.

Fig. 3 presents results for three extended demonstrations. In the first scenario *(a)*, we ask the operator to visit all the corners of the scenario while keeping the robot on the left lane. After arriving back to the first corner, turn back and return following the same instructions. For the second demonstration *(b)*, the operator has to visit the adjacent corners and return, then visit the opposite corner (using the two available paths) and return. For the third demonstration *(c)*, we ask the operator to visit the opposite corner, return, and visit it again using a different path. This behaviour has to be repeated at least once. Top-left image in Fig. 3(a) shows the demonstration path (time from light- to dark-blue colour). The top-right image shows the visiting and transit goals inferred by our system (red and white dots respectively). The bottom images depict the sequence of the goals that are used to infer the program and the blue lines the linear controller that satisfy the transit from one goal to the next. The controllers are clustered and sequenced to build the final program that represents the demonstration. We include the programs for these scenarios in Appendix A. For scenarios *(a)* (App. 3) the induced program can not be simplified and is presented as a list of controllers. For scenario *(b)* (App. 4) two palindromes are found (where the robot turns back) and translated into an iterative loop in the program. For scenario *(c)* (App. 5), the system finds the repetition of the sequence and the palindromes where the robot turns back, adding one level of nested loops. We argue that these programs require a lower extraneous cognitive load (amount of effort placed on the working memory during a task caused by the way the task is presented to the user [2, 65]) for an expert user to understand the demonstration. The program can be examined directly compared to waiting for the demonstration to be executed. Also, the goals are automatically detected by the system rather than by the user. Other ways of quantifying the readability a program are based on the human perception of the quality of the program itself [9, 10]. The quality of the program can be measured by its size, cyclomatic complexity, inter-procedural nesting or abtraction complexity, among others. That type of measure is out of the scope of this work as we are not focused on finding the most readable program, but a suitable translation from a demonstration to a program. In the next section, we show the usefulness of the program abstraction for synthesising new behaviour and for robotic control.

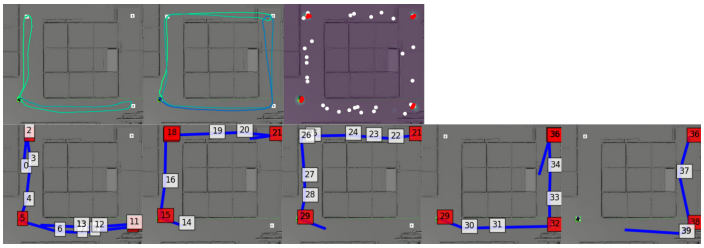
The next section describes how to synthesise new programs and how to run them autonomously on the robot.

4 Program Synthesis and Autonomous Control

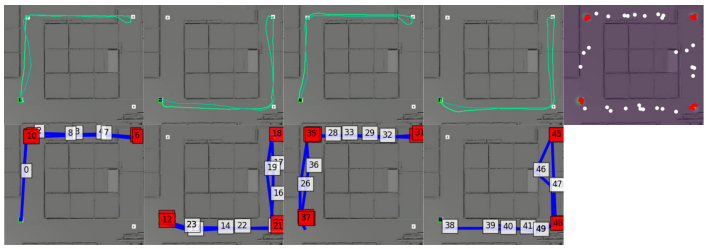
In the previous section, we showed how we inferred goals from a single demonstration. The method also infers proportional controllers that satisfy the demonstration. For robotic control purposes, we would like



(a) Keep on the left side.



(b) Incremental navigation.



(c) Arrive to the other side and return.

Figure 3: Extended surveillance scenarios. For all of them, we show the recorded position of the tele-operated demonstration (top-left images), the inferred visiting and transit goals (red and white dots in the purple-background top-right image). At the bottom of each scenario the sequence of controllers (presented in multiple images). Inferred programs in Appendix A.

to play the controllers back to the robot and have the demonstration autonomously repeated. This type of automation is a core part of learning from demonstration. One of the advantages of the program induction is that such representation of the demonstration allows implementing changes to generalise to new tasks. For example, new visiting orders of the goal areas, and the ease of adding loops and conditional control flow without losing generalisation. Another advantage is that the demonstration (autonomous behaviour in this case) remains interpretable goal-wise.

In practice, if the robot follows the proportional controllers, generalisation to new configurations of the environment is limited (although generalisation made by modifying the program remains). The visiting-goals controllers do not account for obstacles. Thus, those controllers are unable to generalise to new obstacle positions. Even if we include the transit goal controllers, the generalisation remains bound to the degree of modification of the environment. In order to address this, we use the inferred goals to label the data points of the demonstration. With the labelled data, we can train behavioural models that predict the next step position of the robot for each controller. Then, we can use MPC techniques [13] so the robot can autonomously repeat the behaviour. This method has the advantage that visiting goals are invariant between demonstrations of the same task with different obstacles configuration. In the example of Fig. 2, the robot moves from the goal in area 2 to the goal in area 3. As the position of the goals is known, we can label the demonstration trace from the moment that the robot has arrived at the initial goal in area 2 until it reaches the goal in area 3. Note that training a network to predict the future position of the robot without labels, i.e. training several demonstrations with full traces, will result in the inability of the system to learn a meaningful representation when the behaviour is not distinguishable from another. For example, in two different demonstrations, the robot might traverse the same hall but taking a different route after reaching a corner. Our system would be able to separate these two behaviours with different

labels. Without extra information, it not possible to separate these behaviour (same input, different output) with this type or architecture. Other approaches of unsupervised learning have been proposed to tackle this problem [46], using Gaussian mixture models fit using expectation maximisation [19] and variational approaches for switching state space models [23].

We captured a set of demonstrations with the same inspection goal but with different obstacle configurations, including one scene without obstacles. Figure 2 shows two demonstrations, one with and another without obstacles. The system infers the same visiting goals but different transit goals. We use the high-level visiting goals to label the data points within the demonstrations. To account only for obstacles close to the robot, we reduce the top-view of the scene to a 100x100 pixels images centred on the robot. This reduction increases generalisation as now the behaviour is only locally based and does not depend on the configuration of the whole scene. As input to the network, we include the horizontal and vertical linear velocities and the rotation velocity over the horizontal axis of the robot. The output vector $\mathbf{y}_i \in \mathbb{R}^{10}$ includes the position of the robot after 10, 20, 30, and 50 steps (with the simulation step size of 0.1s). We trained one network for each label, for a set of 5 demonstrations with $\sim$11,000 data pairs. Note that a high number of data points is required for the training of the deep models, but a program can be induced from a single demonstration as shown in Section 3. The network included three convolutional layers with 3 kernels each with sizes of $(7,7),(5,5)$ and $(3,3)$ with RELU activation. These layers allow the network to extract features from the images. After the feature layer we concatenate it with the pose of the robot trough a dense layer with a linear activation function (Fig. 4).

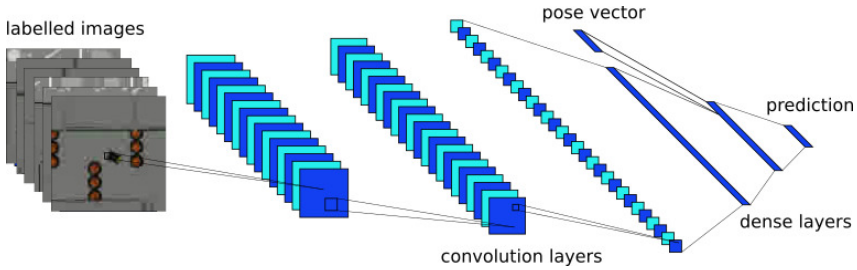


Figure 4: Diagram of the convolutional and dense layers neural network for future pose prediction. Images used as input are a close-up view of the top-view camera with the robot in the centre. The network includes three convolutional layers with a dropout rate of 0.5. The pose information is concatenated to a dense layer output after the last convolution layer. The final output of the network is a fully-connected layer with a linear activation function. We trained a different network for each label, feeding it with data from 5 different demonstrations.

Now, We can use each model to control the robot. The model prediction based controller will calculate a future position of the robot and an internal realisation of a PID controller will drive the robot to the target position. This process is repeated until the goal is reached and the next controller is invoked.

With this set of controllers (one for each model), we can synthesis new programs for behaviours not seen in the demonstration. Listing 2 shows a synthesised program (based on List. 1) where the robot has to arrive from goal 1 to goal 3 traversing goal 2, and then reverse to return to goal 1. This sequence can be extended with the use of conditionals and flow control. Line 2 of the program includes a loop cycle that will repeat the sequence 3 times. Line 5 includes a conditional where at arrival to goal number 3, the system checks if enough time has passed to decide if continuing with the original sequence or call a controller that will drive the robot to goal number 4 and exit the cycle. Fig. 5 shows the diagram of the synthesised program and a run of the program on the robot.

```
TIME_THRESHOLD = 500
for _ in range(1, 3):  # repeat sequence 3 times
    controller(label='goal1_to_goal2')
    controller(label='goal2_to_goal3')
    if timer() > TIME_THRESHOLD:  # check timer
        controller(label='goal3_to_goal4')
        break  # exit loop
    controller(label='goal3_to_goal2')
    controller(label='goal2_to_goal1')
```

Listing 2: The synthesised program, including loop and conditional control flow. The sequence of goals $(1,2,3,2,1)$ is repeated 3 times. A timer is checked if the robot has taken more than a set threshold to reach the third goal. The sequence, cycle and conditionals are not present in any demonstration, but they can be added to the synthesised program and played back by the robot. Fig. 5 shows one loop of the sequence without triggering the conditional.

Even with the possibility to synthesise a more complex program, there are some restrictions to this process. For example, the use of any controller assumes that the actual position of the robot was part of the training of the model. Using a controller outside of its training domain generates undefined behaviour. For the same reason, switching from one controller to a subsequent one can only be synthesised if such switch happened in the demonstration. For example, a controller that drives the robot from goal 2 to goal 1 can only be called if the previous controller finished in goal 2. Even including these restrictions, there are enough combinations to synthesise complex behaviour from a set of few demonstrations.

We build on similar earlier work presented in [11]. In that work, the execution of a same sequence or synthesised plan was restricted to scenarios where all goals were attractors. Here, we also take into account obstacle *avoidance*, i.e., repellers. In our present approach, we have shown how the labelling of the demonstrations and the addition of repellers helps to improve generalisation. Our system is able to train a model with data from different demonstrations. This data represents the same goal (and initial condition) of a subset of the trace, but is executed in different ways and in different scenarios. The MPC scheme takes advantage of these labelling as it can execute in scenarios that are a combination of the ones seen in the demonstrations. Also, the generalisation power of convolutional neural networks improves the complexity of the scenario where the system can run autonomously.

4.1 Learning from Demonstration Comparison

We compare our LfD implementation to [11] as a baseline under the same scenarios. In the baseline, the authors fit a sequence of controllers using sequential importance sampling under a generative switching proportional controller task model. Compared to our system, the goal inference of the baseline does not take into account the distance to the future position of the robot or the saliency of the objects to discriminate between transition and visited goals. Figs. 6(a) and 6(b) show the goals inferred by the baseline system in the scenarios with and without obstacles. The same demonstrations have been used by our system to infer demonstration goals (see Fig. 2). In the scenario without obstacles (Fig. 6(a)), there are goals (yellow dots) outside the path of the robot and also goals over walls. The inferred goals in the scenario with obstacles (Fig. 6(b)) suffers from the same problem, i.e. goals outside a demonstrated path and over objects that have not been visited. When compared to our results, we can see that the inferred goals are closer to the demonstrated paths and are separated between visited (red dots) and transit ones (yellow dots). Our system takes into account the distance between the proposed controllers and the real path of the robot to produce more representative goals of the demonstration. Also, after inference, we

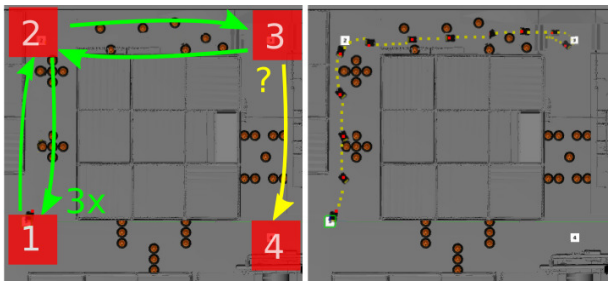
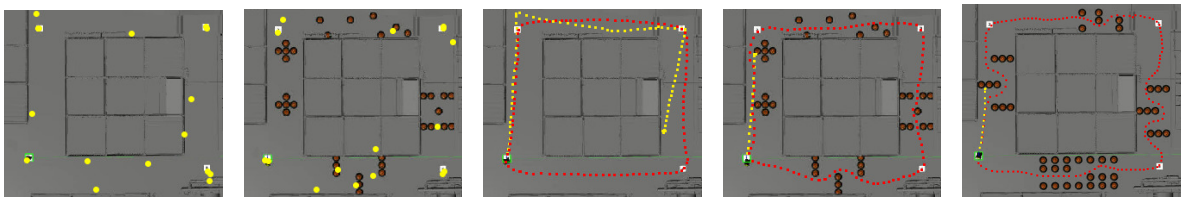


Figure 5: The image on the left depicts the synthesised program from Listing 2. The green arrows are part of the main sequence starting from 1. The yellow arrow represents the conditional path. The red boxes are the areas where the goals are located. The right image shows the synthesised program played back by the robot. The yellow dots are the predicted positions of the robots followed by the MPC. The red dots are the actual position of the robot. The image only shows the trace until line 8 of the code is reached for the first time. In this case, the conditional is not triggered. The remaining repetitions are omitted for clarity.



(a) Baseline (b) Baseline (c) Runs (d) Runs (e) Unseen scenario

Figure 6: Images (a) and (b) show the goals (yellow circles) inferred by the baseline. Images (c) and (d) are runs of autonomous control using the baseline proportional controllers (yellow dots) and our MPC approach (red dots). Image (e) shows the two runs on a scenario where both systems inferred the goals, but the demonstration data was not used to train the behavioural models (only the previous demonstrations). This shows the generalisation capacity of our approach. The baseline system fails to infer goals that are suitable for autonomous control.

include visual grounding of the goals to differentiate between goal types.

We compare the behaviour of the robot under autonomous control. For the baselines, we use the inferred controllers to autonomously drive the robot through the scenario. In our approach, we use the behavioural models learned from the semi-supervised training (Section 4). Figs. 6(c) and 6(d) show the runs by the baseline in yellow, and in red by our system. The baseline follows direct lines between the goals even deviating from the original demonstrations. Also, the baseline is unable to complete a full cycle as the controllers drive the robot to goals that are in the same position as obstacles or walls. To show the generalisation capacity of our approach, we run both approaches in a scenario where goals have been inferred but it has not been used to train the behavioural models. In this case, the behavioural models were trained with demonstration data from previous scenarios. Fig. 6(d) shows that the robot can finish a cycle with our MPC approach (red dots), while the baseline is unable to react to obstacle so the robot fails to arrive at the goal destination.

5 Explanations from Demonstration

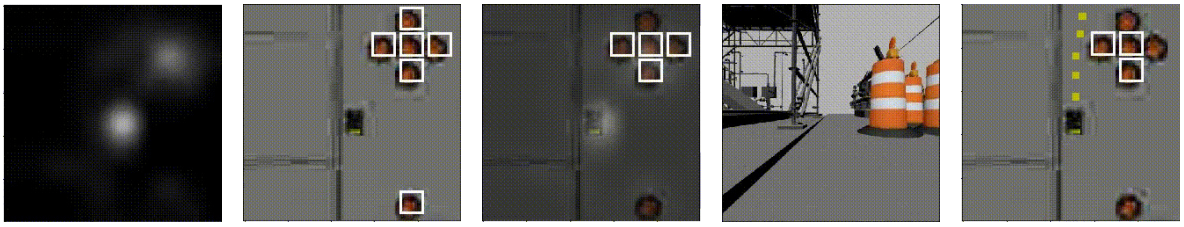
In Section 3, we showed how to extract high-level goals and induce a program from a demonstration. These results already act as an interpretation of the demonstrations. The system abstracts dynamical attractors, modularises and translates the behavioural trace to a language understandable by an operator. An external observer can check the sequence of the demonstration by scrutinising only a few lines of codes instead of observing the full demonstration. Another advantage is that the induced program allows operators to track the inspection process. During the running of the program or by a repetition of the demonstration, the system is capable to establish the actual stage of the inspection, indicating origin and destination goal.

5.1 Causal Analysis

We complement the explanations with black-box analysis [58] of the input in the prediction model $I(\mathbf{x})$. Using causal sensitivity analysis [30] and object classification, we are able to identify the objects that most influence the predictions. In this case, the predictions of future position and rotation of the robot have been learned by the model following the demonstration. For autonomous control (Section 4), this analysis is a direct explanation of the behaviour of the robot. In the case of teleoperation of the robot, this analysis is an educated guess of the objects that influenced the actions chosen by the operator. This post hoc analysis is helpful when there is no access to query the operator directly, but there is access to a recording of the demonstration.

For sensitivity analysis, we modify the input image of a single prediction with a black patch of the same size of the robot in the local top-view camera. We apply a single patch with a stride of half the size of the patch until covering the whole image. For each placement of the patch, we use the Euclidean distance between the initial prediction and the prediction made with the modified image, as the pose prediction are in Euclidean space. After applying the patch in all the positions, we build a normalised saliency heat map (Fig. 7(a)). In the filter, white colour represents blobs of salient pixels. Then, we detect objects in the image (Fig. 7(b)) and filter these with a binary decision based on the saliency in the heat map. We tried several values for the threshold for the binary decision. Among those values, a threshold of 0.8 yielded the best results, in our dataset, to include salient object-based in the size of the white blobs in the heat map (Fig. 7(c)). A lower value for the threshold would add blobs that do not correspond to object, and a higher value would discard all objects as salient. At the next step, we apply a second filter to only represent objects appearing on the on-board camera (Figs. 7(d) and 7(e)). This filter relates to the actual image that the operator perceives in an immersive demonstration. In the virtual environment of the digital twin setup, for practical reasons we assume access to the actual position of the object. However, the system can use widely available computer vision techniques for object detection like supervised learning with convolutional neural networks and 3D SLAM [33, 52]. This is the case for real application including robots deployed in industrial environments. Our approach is modular so implementations like [41,54,55] can be used when there is no direct access to the position of the objects.

The result of the causal analysis (Fig. 7(e)) shows both a prediction of the most likely future behaviour of the robot (yellow dots) and the objects that the controller used to decide the next action (white squares). With this information, the user can have a better understanding of teleological and failed behaviour. This information is useful for designing new environments and new tasks that can be both automatised or human-operated. Also, these analyses can give lights to understand why an undesired action was taken by the robot and in case of failure, reduce the space of possible causes.



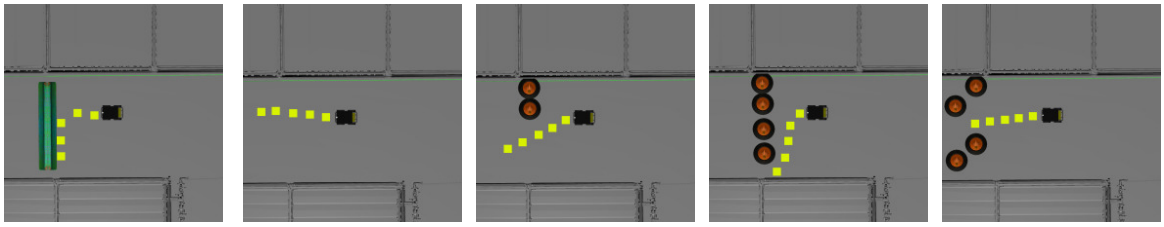
(a) Heat map (b) Detected objects (c) Filtered objects (d) On-board camera (e) Explanation

Figure 7: (a) heat map of image saliency for a model prediction. (b) all objects in the vicinity of the robot. (c) objects filtered by the heat map. (d) on-board camera. (e) final result of applying the filters, including filtering objects that only appear on the on-board camera and the prediction of future position of the robot. The detection of salient objects is an explanation of the objects taken into consideration by either the operator while controlling the robot or by the models while in autonomous control.

5.2 Counterfactuals Explanation

A counterfactual explanation describes the results of hypothetical cases compared to real ones. These characteristics can be a key component for reliability in an autonomous system. Counterfactuals can be used to test the limits of the system. For example, they could be used to find the smallest change to the input of a neural network that changes the classification outcome [67]. In our system, the predictive model can be used for counterfactual causal inference. This inference can help an operator to check the consistency of the system under different scenarios before deployment. For example, in the inspection scenario, the operator can test how sensitive the system is to changes in the position of obstacles. Counterfactuals can also be used to check how the scenario can be modified and still solve the original task. For the inspection task, if the robot fails to arrive at the desired area, the operator can modify the input to the system searching for a configuration that would have allowed the robot to reach the goal.

To test our system, we trained the behavioural model with extra demonstrations where obstacles block the passage of the robot. In these cases, the teleoperator has to find new routes or cancel the operation. In [66, 67], the authors minimise a distance function between the actual state and a modified one. To find the minimum, the authors use exhaustive search over the state space, using high-level features for dimensionality reduction. In our approach, we ask an operator to modify the input image at the pixel level. The modifications include adding and removing objects, and change the position of existing objects. The operator can assess each new configuration and compare the results from the model. Fig. 8(a) shows a failed attempt to advance from left to right as a green barrier is obstructing the passage. During demonstration time, the operator consistently took a U-turn when faced with this obstacle configuration. The model learned this behaviour and correctly predicts the future position of the robot resembling the demonstrations. Now, the operator has the chance to explore a new obstacle configuration by modifying the input image. With each new configuration, the model predicts future positions. For example, in the first modification presented on Fig. 8(b) the barrier has been removed and the behaviour prediction indicates that the robot can continue moving to the left. The other images in Fig. 8(b) show different obstacles placed in front of the robot with their corresponding predictions. All these predictions happen at the model level by modification of the image and do not require changes in the actual environment. This feature is useful when the modification of the environment requires a large effort, for example, in real-life scenarios where moving the obstacles would require human intervention. In these cases, more complex techniques than direct pixel modification are available, e.g. region filling or



(a) Original behaviour (b) Modifications of the original scenario by removing or adding objects.

Figure 8: (a) shows the prediction (yellow dots) of the positions for the robot when a green barrier is in front of it. (b) modifications to the original image by removing the barrier or adding new objects in different positions. The new predictions allow the operator to: i) causally infer that the barrier is responsible for the original behaviour, and ii) find a solution for the robot to continue to the goal.

object removal [16,17]. These techniques can be incorporated into the system to obtain the counterfactual conditionals when a top-view camera with a consistent background is not available.

6 Conclusion

We presented a system for program induction and explainability in learning from demonstration settings in a surveillance scenario with access to proprioceptive and onboard sensor signals. Our approach follows [11], extending it by adding dynamical repellers for obstacles, and a new controller learning process that allows us to synthesise new (obstacle aware) programs that can be successfully run by a robot without external intervention in the presented digital twin scenario.

We have presented a semi-supervised system that abstract goals and proportional controllers from an immersive teleoperated demonstration of an inspection scenario. With this information extracted from the demonstrations, we have been able to induce a program that represents the original demonstration in a programming language for an industrial inspection task.

From the induced program, we were able to extend the demonstrations to new tasks. We showed how new programs can be synthesised and autonomously run on the robots. In a semi-supervised approach, we use the inferred goals within the trace to label the demonstration. This labelling allowed us to train end-to-end models that are used for model predictive control. With the synthesis of programs and the generalisation ability of the deep neural network models, we presented a subset of new programs that where applied to unseen environment. This generalisation has the potential to spawn a wide range of new behaviours and complex tasks from a reduced set of demonstrations.

Also, we presented two approaches on how to use the trained models to explain the demonstrations and autonomous behaviour. These explainable features add post hoc explanations for black-box models, complexity reduction and decomposition of the demonstration. Through the exploitation of the low-level end-to-end models, we have been able to build a more informative system that makes explicit the salient objects and present behavioural counterfactuals.

Our approach used position on the plane as prior for goals. This approach can be generalised to other domains, e.g. 3D spaces or joint angle manipulator control. A task like tower assembly with joint angles as state-space [11] can be extended to include obstacles with our approach. In general, if the goals can be satisfied by proportional controllers then this method can be applied. For more complex behaviour, techniques like Dynamic Movement Primitives [60] can be implemented.

References

[1] D Adiwardana, Akihiro Matsukawa & Jay Whang (2016): *Using generative models for semi-supervised learning*. In: *Medical image computing and computer-assisted intervention–MICCAI*, 2016, pp. 106–14.

[2] Muneeb Imtiaz Ahmad, Jasmin Bernotat, Katrin Lohan & Friederike Eyssel (2019): *Trust and Cognitive Load During Human-Robot Interaction*. *arXiv preprint arXiv:1909.05160*.

[3] Brenna D Argall, Sonia Chernova, Manuela Veloso & Brett Browning (2009): *A survey of robot learning from demonstration*. *Robotics and autonomous systems* 57(5), pp. 469–483, doi:10.1016/j.robot.2008.10.024.

[4] Christopher G Atkeson & Stefan Schaal (1997): *Robot learning from demonstration*. In: *ICML*, 97, Citeseer, pp. 12–20.

[5] Jamshid Bagherzadeh & Hasan Asil (2019): *A review of various semi-supervised learning models with a deep learning and memory approach*. *Iran Journal of Computer Science* 2(2), pp. 65–80, doi:10.1007/s42044-018-00027-6.

[6] Eric Bauer & Ron Kohavi (1999): *An empirical comparison of voting classification algorithms: Bagging, boosting, and variants*. *Machine learning* 36(1-2), pp. 105–139, doi:10.1023/A:1007515423169.

[7] A. Billard & D. Grollman (2013): *Robot learning by demonstration*. *Scholarpedia* 8(12), p. 3824, doi:10.4249/scholarpedia.3824. Revision #138061.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang et al. (2016): *End to end learning for self-driving cars*. *arXiv preprint arXiv:1604.07316*.

[9] Jürgen Börstler, Michael E Caspersen & Marie Nordström (2016): *Beauty and the Beast: on the readability of object-oriented example programs*. *Software quality journal* 24(2), pp. 231–246, doi:10.1007/s11219-015-9267-5.

[10] Jürgen Börstler, Marie Nordström & James H Paterson (2011): *On the quality of examples in introductory Java textbooks*. *ACM Transactions on Computing Education (TOCE)* 11(1), pp. 1–21, doi:10.1145/1921607.1921610.

[11] Michael Burke, Svetlin Penkov & Subramanian Ramamoorthy (2019): *From Explanation to Synthesis: Compositional Program Induction for Learning From Demonstration*. *Robotics: Science and Systems (R:SS)*, doi:10.15607/RSS.2019.XV.015.

[12] Jesse Butterfield, Sarah Osentoski, Graylin Jay & Odest Chadwicke Jenkins (2010): *Learning from demonstration using a multi-valued function regressor for time-series data*. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*, IEEE, pp. 328–333, doi:10.1109/ICHR.2010.5686284.

[13] Eduardo F Camacho & Carlos Bordons Alba (2013): *Model predictive control*. Springer Science & Business Media, doi:10.1007/978-1-4471-3398-8.

[14] Olivier Chapelle, Bernhard Scholkopf & Alexander Zien (2009): *Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]*. *IEEE Transactions on Neural Networks* 20(3), pp. 542–542, doi:10.1109/TNN.2009.2015974.

[15] Silvia Chiappa & Jan R Peters (2010): *Movement extraction by detecting dynamics switches and repetitions*. In: *Advances in neural information processing systems*, pp. 388–396.

[16] Antonio Criminisi, Patrick Perez & Kentaro Toyama (2003): *Object removal by exemplar-based inpainting*. In: *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, 2, IEEE, pp. II–II, doi:10.1109/CVPR.2003.1211538.

[17] Antonio Criminisi, Patrick Pérez & Kentaro Toyama (2004): *Region filling and object removal by exemplar-based image inpainting*. *IEEE Transactions on image processing* 13(9), pp. 1200–1212, doi:10.1109/TIP.2004.833105.

[18] C. Daniel, G. Neumann & J. Peters (2012): *Learning concurrent motor skills in versatile solution spaces*. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3591–3597, doi:10.1109/IROS.2012.6386047.

[19] Arthur P Dempster, Nan M Laird & Donald B Rubin (1977): *Maximum likelihood from incomplete data via the EM algorithm*. *Journal of the Royal Statistical Society: Series B (Methodological)* 39(1), pp. 1–22.

[20] Kevin R Dixon & Pradeep K Khosla (2004): *Trajectory representation using sequenced linear dynamical systems*. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, 4, IEEE, pp. 3925–3930, doi:10.1109/ROBOT.2004.1308881.

[21] Filip Karlo Došilović, Mario Brčić & Nikica Hlupić (2018): *Explainable artificial intelligence: A survey*. In: *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, IEEE, pp. 0210–0215, doi:10.23919/MIPRO.2018.8400040.

[22] Yiwei Fu, Devesh K Jha, Zeyu Zhang, Zhenyuan Yuan & Asok Ray (2019): *Neural Network-Based Learning from Demonstration of an Autonomous Ground Robot*. *Machines* 7(2), p. 24, doi:10.3390/machines7020024.

[23] Zoubin Ghahramani & Geoffrey E Hinton (2000): *Variational learning for switching state-space models*. *Neural computation* 12(4), pp. 831–864, doi:10.1162/089976600300015619.

[24] Andrew B Goldberg & Xiaojin Zhu (2006): *Seeing stars when there aren't many stars: graph-based semi-supervised learning for sentiment categorization*. In: *Proceedings of the first workshop on graph based methods for natural language processing*, Association for Computational Linguistics, pp. 45–52, doi:10.3115/1654758.1654769.

[25] Elena Gribovskaya, S. M. Khansari-Zadeh & Aude Billard (2011): *Learning Nonlinear Multivariate Dynamics of Motion in Robotic Manipulators [accepted]*. *International Journal of Robotics Research* 30(8), pp. 80–117, doi:10.1177/0278364910376251. Available at `http://infoscience.epfl.ch/record/148817`.

[26] D. H. Grollman & O. C. Jenkins (2010): *Incremental learning of subtasks from unsegmented demonstration*. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 261–266, doi:10.1109/IROS.2010.5650500.

[27] Daniel H Grollman & Odest Chadwicke Jenkins (2008): *Sparse incremental learning for interactive robot control policy estimation*. In: *2008 IEEE International Conference on Robotics and Automation*, IEEE, pp. 3315–3320, doi:10.1109/ROBOT.2008.4543716.

[28] Helen Hastie, Katrin Lohan, Mike Chantler, David A Robb, Subramanian Ramamoorthy, Ron Petrick, Sethu Vijayakumar & David Lane (2018): *The ORCA hub: Explainable offshore robotics through intelligent interfaces*. *arXiv preprint arXiv:1803.02100*.

[29] Marco Hutter, Christian Gehring, Andreas Lauber, Fabian Gunther, Carmine Dario Bellicoso, Vassilios Tsounis, Péter Fankhauser, Remo Diethelm, Samuel Bachmann, Michael Blösch et al. (2017): *ANYmal-toward legged robots for harsh environments*. *Advanced Robotics* 31(17), pp. 918–931, doi:10.1080/01691864.2017.1378591.

[30] Jinkyu Kim & John Canny (2017): *Interpretable learning for self-driving cars by visualizing causal attention*. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2942–2950, doi:10.1109/ICCV.2017.320.

[31] Augustine Kong, Jun S Liu & Wing Hung Wong (1994): *Sequential imputations and Bayesian missing data problems*. *Journal of the American statistical association* 89(425), pp. 278–288, doi:10.1080/01621459.1994.10476469.

[32] George Konidaris & Andrew G Barto (2009): *Skill discovery in continuous reinforcement learning domains using skill chaining*. In: *Advances in neural information processing systems*, pp. 1015–1023.

[33] Alex Krizhevsky, Ilya Sutskever & Geoffrey E Hinton (2012): *Imagenet classification with deep convolutional neural networks*. In: *Advances in neural information processing systems*, pp. 1097–1105, doi:10.1145/3065386.

[34] Dana Kulić, Christian Ott, Dongheui Lee, Junichi Ishikawa & Yoshihiko Nakamura (2012): *Incremental Learning of Full Body Motion Primitives and Their Sequencing Through Human Motion Observation*. Int. J. Rob. Res. 31(3), pp. 330–345, doi:10.1177/0278364911426178.

[35] Brenden M Lake, Ruslan Salakhutdinov & Joshua B Tenenbaum (2015): *Human-level concept learning through probabilistic program induction*. Science 350(6266), pp. 1332–1338, doi:10.1126/science.aab3050.

[36] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum & Samuel J Gershman (2017): *Building machines that learn and think like people*. Behavioral and brain sciences 40, doi:10.1017/S0140525X16001837.

[37] Sergey Levine & Pieter Abbeel (2014): *Learning neural network policies with guided policy search under unknown dynamics*. In: Advances in Neural Information Processing Systems, pp. 1071–1079.

[38] Sergey Levine, Chelsea Finn, Trevor Darrell & Pieter Abbeel (2016): *End-to-end training of deep visuomotor policies*. The Journal of Machine Learning Research 17(1), pp. 1334–1373.

[39] Fangtao Huang Li, Minlie Huang, Yi Yang & Xiaoyan Zhu (2011): *Learning to identify review spam*. In: Twenty-second international joint conference on artificial intelligence.

[40] Zachary C Lipton (2018): *The mythos of model interpretability*. Queue 16(3), pp. 31–57, doi:10.1145/3233231.

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu & Alexander C Berg (2016): *Ssd: Single shot multibox detector*. In: European conference on computer vision, Springer, pp. 21–37, doi:10.1007/978-3-319-46448-0_2.

[42] Yin Lou, Rich Caruana & Johannes Gehrke (2012): *Intelligible models for classification and regression*. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 150–158, doi:10.1145/2339530.2339556.

[43] Laurens van der Maaten & Geoffrey Hinton (2008): *Visualizing data using t-SNE*. Journal of machine learning research 9(Nov), pp. 2579–2605.

[44] Olivier Mangin & Pierre-Yves Oudeyer (2011): *Unsupervised learning of simultaneous motor primitives through imitation*. In: IEEE ICDL-EPIROB 2011, Frankfurt, Germany. Available at `https://hal.archives-ouvertes.fr/hal-00652346`.

[45] Gary F Marcus (2018): *The algebraic mind: Integrating connectionism and cognitive science*. MIT press, doi:10.7551/mitpress/1187.001.0001.

[46] Roderick Murray-Smith & T Johansen (1997): *Multiple model approaches to nonlinear modelling and control*. CRC press.

[47] Andrew Y Ng, Stuart J Russell et al. (2000): *Algorithms for inverse reinforcement learning*. In: Icml, 1, p. 2.

[48] Èric Pairet, Paola Ardón, Xingkun Liu, José Lopes, Helen Hastie & Katrin S Lohan (2019): *A Digital Twin for Human-Robot Interaction*. In: 2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI), IEEE, pp. 372–372, doi:10.1109/HRI.2019.8673015.

[49] Peter Pastor, Heiko Hoffmann, Tamim Asfour & Stefan Schaal (2009): *Learning and generalization of motor skills by learning from demonstration*. In: 2009 IEEE International Conference on Robotics and Automation, IEEE, pp. 763–768, doi:10.1109/ROBOT.2009.5152385.

[50] Svetlin Penkov & Subramanian Ramamoorthy (2017): *Using program induction to interpret transition system dynamics*. arXiv preprint arXiv:1708.00376.

[51] Lerrel Pinto & Abhinav Gupta (2016): *Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours*. In: 2016 IEEE international conference on robotics and automation (ICRA), IEEE, pp. 3406–3413, doi:10.1109/ICRA.2016.7487517.

[52] Charles R Qi, Hao Su, Kaichun Mo & Leonidas J Guibas (2017): *Pointnet: Deep learning on point sets for 3d classification and segmentation*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 652–660.

[53] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov & Sergey Levine (2017): *Learning complex dexterous manipulation with deep reinforcement learning and demonstrations*. arXiv preprint arXiv:1709.10087, doi:10.15607/RSS.2018.XIV.049.

[54] Joseph Redmon & Ali Farhadi (2018): *Yolov3: An incremental improvement*. arXiv preprint arXiv:1804.02767.

[55] Shaoqing Ren, Kaiming He, Ross Girshick & Jian Sun (2015): *Faster r-cnn: Towards real-time object detection with region proposal networks*. In: Advances in neural information processing systems, pp. 91–99, doi:10.1109/TPAMI.2016.2577031.

[56] Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin (2016): *" Why should I trust you?" Explaining the predictions of any classifier*. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 1135–1144, doi:10.18653/v1/N16-3020.

[57] Clearpath Robotics (2014): *Husky, unmanned ground vehicle*. Available at `https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot`.

[58] Wojciech Samek, Thomas Wiegand & Klaus-Robert Müller (2017): *Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models*. arXiv preprint arXiv:1708.08296.

[59] Makoto Sato & Hiroshi Tsukimoto (2001): *Rule extraction from neural networks via decision tree induction*. In: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222), 3, IEEE, pp. 1870–1875, doi:10.1109/IJCNN.2001.938448.

[60] Stefan Schaal (2006): *Dynamic movement primitives-a framework for motor control in humans and humanoid robotics*. In: Adaptive motion of animals and machines, Springer, pp. 261–280, doi:10.1007/4-431-31381-8_23.

[61] Stefan Schaal, Jan Peters, Jun Nakanishi & Auke Ijspeert (2005): *Learning movement primitives*. In: Robotics research. the eleventh international symposium, Springer, pp. 561–572, doi:10.1007/11008941_60.

[62] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh & Dhruv Batra (2017): *Grad-cam: Visual explanations from deep networks via gradient-based localization*. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 618–626, doi:10.1007/s11263-019-01228-7.

[63] Karen Simonyan, Andrea Vedaldi & Andrew Zisserman (2013): *Deep inside convolutional networks: Visualising image classification models and saliency maps*. arXiv preprint arXiv:1312.6034.

[64] Jost Tobias Springenberg (2015): *Unsupervised and semi-supervised learning with categorical generative adversarial networks*. arXiv preprint arXiv:1511.06390.

[65] John Sweller (2011): *Cognitive load theory*. In: Psychology of learning and motivation, 55, Elsevier, pp. 37–76, doi:10.1007/978-1-4419-1428-6_446.

[66] Arnaud Van Looveren & Janis Klaise (2019): *Interpretable counterfactual explanations guided by prototypes*. arXiv preprint arXiv:1907.02584.

[67] Sandra Wachter, Brent Mittelstadt & Chris Russell (2017): *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GPDR*. Harv. JL & Tech. 31, p. 841, doi:10.2139/ssrn.3063289.

[68] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot & Nando Freitas (2016): *Dueling network architectures for deep reinforcement learning*. In: International conference on machine learning, pp. 1995–2003.

[69] Yaochu Jin (2000): *Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement*. IEEE Transactions on Fuzzy Systems 8(2), pp. 212–221, doi:10.1109/91.842154.

[70] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel & Sergey Levine (2018): *One-shot imitation from observing humans via domain-adaptive meta-learning*. arXiv preprint arXiv:1802.01557, doi:10.15607/RSS.2018.XIV.002.

[71] Matthew D Zeiler & Rob Fergus (2014): *Visualizing and understanding convolutional networks*. In: *European conference on computer vision*, Springer, pp. 818–833, doi:10.1007/978-3-319-10590-1_53.

[72] Xiaojin Jerry Zhu (2005): *Semi-supervised learning literature survey*. Technical Report, University of Wisconsin-Madison Department of Computer Sciences.

[73] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas et al. (2018): *Reinforcement and imitation learning for diverse visuomotor skills*. *arXiv preprint arXiv:1802.09564*, doi:10.15607/RSS.2018.XIV.009.

[74] Jan Ruben Zilke, Eneldo Loza Mencía & Frederik Janssen (2016): *Deepred–rule extraction from deep neural networks*. In: *International Conference on Discovery Science*, Springer, pp. 457–473, doi:10.1007/978-3-319-46307-0_29.

A Inferred programs for extended scenarios

Programs for the extended scenarios presented in Fig. 3. The command *execute(goal_label)* calls the controller to arrive from the previous goal to the one defined as parameter. If there is no previous goal, the origin is assumed. Goal number correspond to position as presented in Fig. 2(b). The labels with cardinal direction (1W, 1E, 2N, 2S, 3W, 3E, 4N and 4S) represent the goals in the middle of the halls.

```
def program():
  execute(1)
  execute(1W)
  execute(2)
  execute(2N)
  execute(3)
  execute(3E)
  execute(4)
  execute(4S)
  execute(1)
  execute(4N)
  execute(4)
  execute(3W)
  execute(3)
  execute(2S)
  execute(2)
  execute(1E)
  execute(1)
  return
```

Listing 3: Extended scenario (a).

```
def program():
  controller_list = [1,2]
  count = 0
  for k in range(len(controller_list)*2-1):
    execute(controller_list[count])
    if k >= len(controller_list)-1:
      count = count-1
    else:
      count = count+1
  controller_list = [4,1,2,3]
  count = 0
  for k in range(len(controller_list)*2-1):
    execute(controller_list[count])
    if k >= len(controller_list)-1:
      count = count-1
    else:
      count = count+1
  execute(3)
  execute(4)
  execute(1)
  return
```

Listing 4: Extended scenario (b).

```
def program():
  for j in range(2):
    controller_list = [1,2,3]
    count = 0
    for k in range(len(controller_list)*2-1):
      execute(controller_list[count])
      if k >= len(controller_list)-1:
        count = count-1
      else:
        count = count+1
    controller_list = [4,3]
    count = 0
    for k in range(len(controller_list)*2-1):
      execute(controller_list[count])
      if k >= len(controller_list)-1:
        count = count-1
      else:
        count = count+1
  execute(1)
  return
```

Listing 5: Extended scenario (c).