ETH zürich

ETH Library

A Bayesian tracker for synthesizing mobile robot behaviour from demonstration

Journal Article

Author(s):
Magnenat, Stéphane; Colas, Francis

Publication date:
2021-12

Permanent link:
https://doi.org/10.3929/ethz-b-000513240

Rights / license:
In Copyright - Non-Commercial Use Permitted

Originally published in:
Autonomous Robots 45, https://doi.org/10.1007/s10514-021-10019-4

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

Noname manuscript No.
(will be inserted by the editor)

A Bayesian Tracker for Synthesizing Mobile Robot Behaviour from Demonstration

Stéphane Magnenat · Francis Colas

the date of receipt and acceptance should be inserted later

Abstract Programming robots often involves expert knowledge in both the robot itself and the task to execute. An alternative to direct programming is for a human to show examples of the task execution and have the robot perform the task based on these examples, in a scheme known as learning or programming from demonstration. We propose and study a generic and simple learning-from-demonstration framework. Our approach is to combine the demonstrated commands according to the similarity between the demonstrated sensory trajectories and the current replay trajectory. This tracking is solely performed based on sensor values and time and completely dispenses with the usually expensive step of precomputing an internal model of the task. We analyse the behaviour of the proposed model in several simulated conditions and test it on two different robotic platforms. We show that it can reproduce different capabilities with a limited number of meta parameters.

Keywords programming by demonstration, learning from demonstration, non-parametric Bayesian model, teach and repeat, online tracking

1 Introduction

A large variety of mobile robots has blossomed recently, made possible by progresses in energy storage, electronics, processing speed, and other technological improvements. We have seen these robots performing exploration and navigation tasks in different environments, and demonstrating impressive autonomy. However, the actual programming of complex behaviours remains tedious, in particular when physical interaction takes place between the robot and its environment. This process often involves writing and tuning complicated state machines and requires a lot of expert engineering resources. Machine learning, which aims at specifying complex algorithms by leveraging data instead of just expert knowledge, proposes an alternative path. It appears especially useful for robotic tasks, as these often deal with high-dimensional input and output spaces while relying on a lot of contingent parameter values. The specification of robot behaviours from recorded training data forms the field of *programming by* or *learning from demonstration* as described in the survey from Argall et al. (2009) or in chapter 74 of the Handbook of Robotics by Siciliano and Khatib (2016).

While this field ultimately aims at finding a solution to program any robotic task on a large variety of robotic platforms, currently the choice of the approach mostly depends on both the platform and the kind of tasks to be executed. Research has explored complex problems in a variety of directions. In particular, many recent works tackle the transfer of a skill from one space to another, or the teaching of robots with a high-dimensional actuator space. However, solving these problems requires expert knowledge to select, tune, and deploy the right algorithm in a given field and for a given task. Often, these works do not propose a unified approach. In this paper, we are interested in relatively simple robots that can be tele-operated, and hence we do not explore the problem of the difference in representation between the demonstration and replay data. Moreover, we do not explore the problem of explicit dimensionality reduction in the

S. Magnenat
Game Technology Center, Computer Science Dpt, ETH Zürich

F. Colas
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
E-mail: francis.colas@inria.fr
Corresponding author

sensor or actuator space. This excludes high-dimensional systems such as humanoids.

Specifically, the present work aims at understanding, with the requirement of easy deployment on a wide range of platforms, the limits of capabilities that can be achieved with a simple, non-parametric system that learns from demonstration. We want the system to be usable by non-specialists, for example a robotics technician, rather than only by the developers of the algorithm. This requires to minimise the number of meta parameters, because these ultimately have to be given to the system in addition to the demonstrations. To study such a system, we first propose a taxonomy of the capabilities for an autonomous and adaptive robot. Formally, the robot control algorithm should be able to:

– **Cope with noise in the sensors**. Sensors are always noisy and therefore the perceived values at execution time will never be exactly similar to the ones at learning time. Therefore, a noise model must be specified per sensor. Moreover, there might be outliers, that is, values irrelevant to the task, for example sun glare. These outliers, if not present on all sensors at the same time, should not fully confuse the execution of the robot.
– **Consider timing to trigger action**. Sometimes, a robot might have to do an action at a specific time after a given observation, in the form of open-loop control. For example, a robot operating in a factory might have to learn about delays in a processing chain.
– **Take action in function of sensory inputs**. The robot should not blindly replay a recorded trajectory but should be able to react to sensory events such as avoiding an upcoming obstacle. This capability is often referred to as being reactive. For example, if a mobile robot is trained to follow a wall either at its left or its right, its actions should depend on the presence of the wall at a given side.
– **Consider past observations to decide of future actions**. Sometimes the right action to do at a given time depends on the past observations. For example, a robot might have to turn left or right at a T-junction in function of some past observation. This requires having a memory and can correspond to state machines.

These capabilities are sometimes in competition, for example a perfectly reactive robot will have no memory, and a robot basing its action solely on memory will be a pure automaton.

This paper proposes a minimalist white-box system with only $n_s + 3$ meta parameters, where n_s is the number of sensor dimensions. All these parameters are related to the robotic platform and its application, but do not depend on the task.

Our system, albeit simple, can reproduce a trajectory considering noisy sensors; it can trigger actions by time, and is able to keep memory of past events to decide future actions. The different capabilities are controlled by the 3 meta parameters not linked to the sensors. We analyse the choice of these in simulation. Then, we demonstrate our system on a miniature research robot (marXbot) performing manipulation, and on a full-scale field platform (artor) performing navigation. Finally, we discuss the generality of our approach and lay out future research steps.

2 Related Work

There have been mainly three approaches to learning from demonstration:

– learning a mapping between configuration and commands,
– learning a mapping between sensors and actuators,
– and learning a task by splitting it into a sequence of motion primitives.

2.1 Mapping configuration to command

A major application of learning from demonstration can be found in pick-and-place or other manipulation tasks with a robotic arm. Indeed, the configuration space of a robotic arm is very different from the workspace and is usually of high dimension, which makes it difficult to reason on for a non-expert. A large variety of algorithms have been devised to learn a non-linear mapping between the configuration and actuators.

Classical works in this field, such as by Calinon et al. (2007), use Gaussian models to do regression and generalisation between several recorded trajectories. Recent development in this field include mixture models using different reference frames (Calinon 2016) or even semi-Markov models for better temporal evolutions (Havoutis and Calinon 2019). This approach requires the specification of several key parameters, such as the number of Gaussian distributions and, for task-parametrized variants, all the relevant frames of the task. Other works in this direction, such as reported by Chatzis et al. (2012) try to circumvent this requirement by using higher-level, non-parametric models such as Dirichlet Processes. However, these models also require the specification of more abstract meta parameters, but hope that these are less dependant on the task. Another difficulty is the choice of the space in which the data should be encoded (for example joint space, end-effector Cartesian space, torque space) and what kind of motion (cyclic or discrete) they should represent.

Another approach to model the motion is to consider dynamical systems (Khansari-Zadeh and Billard 2011; Pastor et al. 2013). For instance, Pastor et al. (2013) have described skills as a combination of dynamic movement primitives. While the attractor dynamics adds smoothing and local adaptivity, this approach still depends on several arbitrary parameters, such as the number of basis functions. For such a system to be able to cope with unexpected elements such as obstacles, its function must be modified explicitly by hand: handling obstacle is not learned. In that sense, dynamic movement primitives do not naturally consider exteroceptive information.

While these approaches have shown impressive results, they cannot be applied when the configuration of the robot is not directly available. Furthermore, as they do not consider sensor values, they are unsuitable for tasks requiring a decision based on some perceived element of the environment.

2.2 Mapping sensors to actuators

The lack of concern with exteroceptive sensors can be overcome by adding a perception layer in front of the pipeline. For instance, Yang et al. (2020) use a deep convolutional neural network to learn high-level features. Then, they use random forests to control an autonomous wheel loader.

In the same vein, one can combine deep learning with dynamic movement primitives in order to do human motion classification and generation. For instance, Chen et al. (2016) train a variational auto-encoder to learn a compressed representation of the sensory space in which they apply a dynamic movement primitive model. The forcing term shaping the motion is also learned using a neural network. A similar structure, in which the dynamic movement primitive constraints are replaced by a dynamic Bayesian network has been reported by Karl et al. (2017). Recently, DelPreto et al. (2020) have leveraged both virtual reality and neural networks to implement an apprenticeship model combining self-supervised learning and human demonstration. These works show the ability to learn, reproduce and recognise various human motions.

Deep learning also opens the possibility of tackling the learning problem in an end-to-end manner. A recent step in this direction is SA-Net (Soans et al. 2020), which is able to recognize state-action pairs using an RGB-D sensor. However, it still needs to be coupled with a policy learning algorithm. Moreover, deep learning usually requires a lot of training data and still lacks a methodology to devise the hyper parameters (e.g. number and type of layers) for a given task.

2.3 Splitting sequences

Approaches presented so far focus on the continuity of the motion and are not well suited for tasks whose natural structures are discrete sequences of steps. In particular, they cannot remember a sensor event in the early part of a task and alter action in function in a later stage.

On the contrary, several approaches, such as proposed by Mühlig et al. (2012), split training sequences into motion primitives by specifying either a structure for the task or a function deciding on a step change, typically looking for discontinuities. The same approach can be used with a mobile manipulator (Konidaris et al. 2012), while driving an autonomous car (Maye et al. 2011), or with a robotic arm (Wu and Kofman 2008). Often, generic segmentation (Keogh et al. 2004) is used. For instance, Loula et al. (2020) learn modes, which are trajectory optimized during replay with TrajOpt (Schulman et al. 2014). Sometimes, ad-hoc information is used like contact points as by Dong and Williams (2012).

While these approaches can handle robotic tasks involving several steps, they require a definition, either explicit or implicit, of those steps. As a consequence, they need a lot of parameters that depend both on the robotic system and on the task. Works such as reported by Grollman and Jenkins (2010) have tackled the question of reducing this number of parameters, for instance by trying to learn both transitions and states in an unsupervised way. However, the latter study concludes that this was not yet possible.

Another way of splitting is by probabilistic models. Daniel et al. (2012) proposes to use a mixture of motion primitive combined through a hidden *option* variable. While the number of motion primitive and their shapes are specified by hand, their parameters and the relation between the states (option values) are learnt. In this work, there is no possibility of sequencing, the choice of the primitive is only controlled by the state (position and speed in the examples presented in the paper). Another approach by Infantes et al. (2011) trains a pre-defined dynamic Bayesian network containing two hidden option variables. That system can potentially recover some sequencing, at the expense of being specific to a robot sensorimotor configuration and a type of task (navigation). Recently, sequencing has been achieved at the symbolic level using influence diagrams and Bayesian networks (Koenig and Matarić 2017). While a step forward, such approach still requires the user to explicitly define task parts.

This question of selecting a subsequence also appears in visual teach and repeat systems (Furgale and Barfoot 2010). The aim is to demonstrate and then reproduce a path with a mobile robot. The path is split into local maps and a key question is to choose the relevant one into which localising the robot. Recent approaches use recommendation techniques (collaborative filtering) for this selection, as MacTavish et al. (2018), or implicit encoding through a neural network, as Xie et al. (2020). However, they use this information to solve the geometric localisation problem before applying standard trajectory planning and following techniques. Therefore, their system is restricted to a very specific application with heavy use of prior knowledge.

A promising approach for imitation learning that surfaced in the last two decades is *inverse reinforcement learning* (Russell 1998; Ng and Russell 2000), for example used with robots by Abbeel and Ng (2004); Abbeel et al. (2010). This approach aims at recovering a reward function from a set of demonstrations. Then, with this reward function, an optimal policy can be computed.

This formulation is typically done using Markov Decision Processes (MDP) and, as such, the state space, the action space and the transition model (the probability of reaching a future state given the current state and the action) are supposed to be known. In order to apply it to a mobile robot, one can try to compute features from the sensory space to enhance the state space. Vasquez et al. (2013) has shown that selecting the features is both important and challenging and, therefore, heavily depends on the task. This approach cannot be used in our case where we do not want to have to specify a state space and estimate the current hidden state from the sequence of observations.

In this paper, we present an approach that learns both basic trajectory elements and their sequencing in an implicit way. While not learning full state machines, our approach can branch between different recorded trajectory parts based on observation. It can both learn timing and adapt the replay speed in function of the environment. We achieve this feature set by using a Bayesian approach that models the replay as a tracking process with respect to the recorded data.

3 Model

Our approach does not try to build a synthetic representation of the training data. On the contrary, the algorithm aims at tracking, in the training data, the most relevant information with respect to the current attempt to reproduce the behaviour. More precisely, we build a Bayesian filter in which we assume that both sensor readings and motor commands are conditioned by trajectory and time indices. In this model, replaying is done by inferring the motor commands by marginalisation over those trajectory and time indices. The model also assumes that motor commands and sensor observations are available at each time step at a certain constant frequency. This model builds on the work of Pradalier and Bessière (2004), adding multiple trajectories. A preliminary version was published in (Magnenat et al. 2012b).

Our model is parameterised by two kinds of values: *meta parameters*, that define the properties of the distributions and depend on the task; and *implementation constants*, that can be kept fixed over a large variety of systems.

3.1 Variables

The formal expression of this model involves defining several variables:

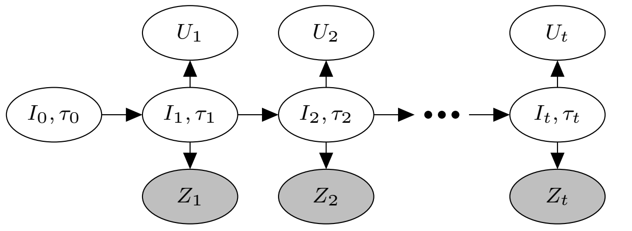
- $\mathit{\Pi} = \{\zeta_t^i, \upsilon_t^i | \forall i \in (1, N), \forall t \in (1, L_i)\}$ Records of N trajectories of lengths $\{L_1, L_2, \ldots, L_N\}$, where trajectory i, at record time step t, has sensor data ζ_t^i and actuator command υ_t^i (vector values). All subsequent formulas are assumed to be conditioned by $\mathit{\Pi}$.
- I_t Index of trajectory at replay time t, ranges from 1 to N.
- τ_t Position on trajectory at replay time t, ranges from 1 to $\max_i L_i$.
- U_t Actuator command at replay time t, vector value.
- Z_t Observation (sensor data) at replay time t, vector value of n_s dimension.

During recording, data are sampled at a frequency C_s. Similarly, at replay time, actuator commands are generated at the same frequency. In our experiments, we fixed C_s to 10 Hz.

3.2 Distributions

We decompose the joint distribution over those variables by leveraging conditional independence assumptions. This leads to a recursive expression of the inference similar to a Bayesian filter (see Figure 1 for the dynamic Bayesian network). The distributions involved are:

- $p(U_t | I_t, \tau_t)$: related to training data,
- $p(Z_t | I_t, \tau_t)$: an observation model,
- $p(I_t | I_{t-1})$: transition between trajectories,



$$p(U_{1:t}, Z_{1:t}, I_{1:t}, \tau_{1:t}) = p(U_{1:t-1}, Z_{1:t-1}, I_{1:t-1}, \tau_{1:t-1})$$
$$p(U_t|I_t, \tau_t)p(Z_t|I_t, \tau_t)p(I_t|I_{t-1})p(\tau_t|\tau_{t-1}) \quad (1)$$

Fig. 1 The graphical representation of the model and the corresponding decomposition.

– $p(\tau_t|\tau_{t-1})$: transition from a time step to the next,
– $p(I_{t-1}, \tau_{t-1}|Z_{1:t-1})$: result of previous inference step.

Note that $p(U_t|I_t, \tau_t)$ is not used directly in the inference, as explained in Section 3.3.2.

3.2.1 Meta parameters.

Our model has only $n_s + 3$ meta parameters: one σ_k per sensor dimension k and 3 global ones θ_I, θ_τ, and α. The parameter σ_k is related to the scale of variation: it basically states when two values are different. It is important to note that this parameter is only lower-bounded by the noise of the sensor but is mostly governed by the semantics of the values. This means that for two sensors adequately measuring the same quantity, this value would be the same even if one sensor is less noisy than the other. For example, for an outdoor wheeled robot, both differential GPS and odometry give an information on the pose, but with a different precision. In that case, if the task is just to reach a large area, the scale factor can be in the order of the meter for both sensors, even if the d-GPS can achieve better precision. Hence while the σ_k parameters might vary for different application contexts, they would be similar for different tasks within the same context, such as reaching different areas in the preceding example. The global parameters θ_I and θ_τ control transitions while α is the probability of measuring an outlier value on a sensor.

To improve readability, in the rest of this document, we will often omit the term *meta* when talking about these parameters.

3.2.2 Observation model.

The observation model is not trivial, in particular because two successive data points can be as far away as 90 % of the space. Therefore, a simple Gaussian modelling of sensor noise situated on the data points is not enough, as it would lead to infinitesimal probabilities when two successive data points are very far away. To cope with this, we consider that the observations are sampled from a piecewise linear function. Thus, we convolve our observation model with both segments in observation space linking observations at time $t-1$ and t, and t and $t+1$.

Moreover, when two points are far away, for the observation, we do not wish to consider the farther away to be significantly less probable than the closest, because anyway they are both wrong. Therefore, we define a robust observation model by a mixture of a uniform on the bounded observation interval and the actual observation function, a Gaussian. This mixture is controlled by α.

By linearity, it is easy to show that this convolved model of a mixture is the mixture of the convolution and a uniform:

$$p(Z_t|I_t = i, \tau_t = j) = \prod_k \Bigg[(1-\alpha) \left(\frac{1}{2} \int_{\zeta^k_{i,j-1}}^{\zeta^k_{i,j}} \frac{1}{\Delta\zeta^k_{i,j-1}} \mathcal{F}(t)dt + \frac{1}{2} \int_{\zeta^k_{i,j}}^{\zeta^k_{i,j+1}} \frac{1}{\Delta\zeta^k_{i,j}} \mathcal{F}(t)dt \right) + \alpha \frac{1}{R^k} \Bigg] \quad (2)$$

where $\Delta\zeta^k_{i,j} = \zeta^k_{i,j+1} - \zeta^k_{i,j}$, $\mathcal{F}(t)$ is the observation function, and R^k is the range of sensor k.

Therefore, for a given sensor k, the integrals can be implemented efficiently using the `erf` function:

$$\begin{aligned}
&\frac{1}{2} \int_{\zeta_{i,j-1}}^{\zeta_{i,j}} \frac{1}{\Delta\zeta_{i,j-1}} \mathcal{N}(t, \sigma^2)dt + \frac{1}{2} \int_{\zeta_{i,j}}^{\zeta_{i,j+1}} \frac{1}{\Delta\zeta_{i,j}} \mathcal{N}(t, \sigma^2)dt \\
=&\frac{1}{2(\zeta_{i,j} - \zeta_{i,j-1})} \Big[C(x, \zeta_{i,j}, \sigma) - C(x, \zeta_{i,j-1}, \sigma) \Big] + \\
&\frac{1}{2(\zeta_{i,j+1} - \zeta_{i,j})} \Big[C(x, \zeta_{i,j+1}, \sigma) - C(x, \zeta_{i,j}, \sigma) \Big] \quad (3)\\
=&\frac{1}{4(\zeta_{i,j} - \zeta_{i,j-1})} \left[\mathrm{erf}\left(\frac{x - \zeta_{i,j}}{\sqrt{2\sigma^2}} \right) - \mathrm{erf}\left(\frac{x - \zeta_{i,j-1}}{\sqrt{2\sigma^2}} \right) \right] + \\
&\frac{1}{4(\zeta_{i,j+1} - \zeta_{i,j})} \left[\mathrm{erf}\left(\frac{x - \zeta_{i,j+1}}{\sqrt{2\sigma^2}} \right) - \mathrm{erf}\left(\frac{x - \zeta_{i,j}}{\sqrt{2\sigma^2}} \right) \right]
\end{aligned}$$

where σ is the amount of sensor noise, $C(x, \mu, \sigma)$ the cumulative distribution function of the Gaussian of mean μ and variance σ^2 evaluated at x, and erf() the error function.

3.2.3 Transition model.

The transition model is structured in two parts: the index of the trajectory and the time position in a given trajectory. These are represented by conditional probability tables. In order to allow smooth-enough transitions in $p(\tau_t|\tau_{t-1})$, we over-sample the resolution on $p(\tau)$ by a factor C_r. When dealing with recorded sensor values and motor commands, we simply expand these data by a factor C_r, with no interpolation. In our experiments, we fixed C_r to 3.

The distribution over the next trajectory index given the past trajectory index is close to an identity matrix but with uniform probability θ_I to jump from one trajectory to another. This ensures a strictly positive lower bound on the probability of each trajectory, which is useful to allow a trajectory that differs from the first part of the observations to meaningfully contribute to the motor commands when the observations start matching:

$$p(I_t|I_{t-1}) = \begin{cases} 1-\theta_I \text{ if } I_t = I_{t-1} \\ \frac{\theta_I}{N-1} \text{ otherwise} \end{cases} \quad (4)$$

The transition for the position inside a given trajectory expresses that this position index most likely gets increased by $1 \times C_r$ but can also change by more or less, with a probability controlled by θ_τ. This allows for slight stretching of time for the replay, according to the observations. To have a symmetric behaviour between time extension and compression, we use a log-normal function to express this relation:

$$p(\tau_t = a|\tau_{t-1} = b) = \begin{cases} \ln\mathcal{N}\left(\frac{a-b}{C_r}, \theta_\tau\right) \text{ if } a > b \\ 0 \text{ otherwise} \end{cases} \quad (5)$$

This can be pre-computed and stored into a sliding window of length W, which, for performance reasons (see Section 3.4), we cut when the value of the function is lower than $C_w = 0.02$ times the value of its mode. Hence, W is found by solving the following equation:

$$\ln\mathcal{N}(e^{-\theta_\tau^2}, \theta_\tau^2) \cdot C_w = \ln\mathcal{N}(\frac{W}{C_r}, \theta_\tau^2) \quad (6)$$

3.2.4 Initial conditions.

The initial condition is a uniform distribution over the trajectories and a Dirac delta function on the first time step:

$$p(I_0 = i, \tau_0 = j) = \begin{cases} 1/N \text{ if } j = 0 \\ 0 \text{ otherwise} \end{cases} \quad (7)$$

3.2.5 Termination criterion.

The task is considered completed if the probability mass is mostly on the last indices of the trajectories. Indeed, contrary to a deterministic automaton, we cannot expect to be sure to be in the terminal state. Moreover, as there are various tracks of various length and we maintain a lower bound on the probability of all trajectories, the probability threshold should not be too high. In the present paper, we used: $p(\tau_t$ in last C_f time steps$) > C_{fp} = 0.9$ with $C_f = 10$.

3.3 Questions

The inference can be divided into three different questions:

- update due to time, involving the prediction model;
- generation of a command, involving a decision function;
- update due to observations, involving the observation model.

This order allows commands to be triggered by time rather than by observation. Indeed, if a change in observation depends on a specific action, the replay needs to actually make this action for the sequence to move on.

3.3.1 Prediction update.

The prediction update applies the transition models for time and trajectory indices. It corresponds to the following inference:

$$\begin{aligned} & p(I_t, \tau_t|Z_{1:t-1}) \\ = & \sum_{I_{t-1},\tau_{t-1}} p(I_t, \tau_t|I_{t-1}, \tau_{t-1}) p(I_{t-1}, \tau_{t-1}|Z_{1:t-1}) \end{aligned}$$

This inference can be simplified according to the underlying structure of $P(I_{t-1}, \tau_{t-1}|Z_{1:t-1})$. Indeed, it can be written as the product $P(I_{t-1}|Z_{1:t-1})P(\tau_{t-1}|Z_{1:t-1})$ in which each factor can be updated separately:

$$\begin{aligned} & p(I_t, \tau_t|Z_{1:t-1}) \\ = & \sum_{I_{t-1}} p(I_t|I_{t-1}) p(I_{t-1}|Z_{1:t-1}) \cdot \\ & \sum_{\tau_{t-1}} p(\tau_t|\tau_{t-1}) p(\tau_{t-1}|I_{t-1}, Z_{1:t-1}) \end{aligned} \quad (8)$$

3.3.2 Getting command U_t at time t.

The probability of a given command U_t is the marginalisation over trajectory and time indices:

$$p(U_t|Z_{1:t-1}) = \sum_{I_t,\tau_t} p(U_t|I_t, \tau_t) p(I_t, \tau_t|Z_{1:t-1}) \quad (9)$$

This expression depends on the prediction update computed above and on $p(U_t|I_t, \tau_t)$, which is not known. However, if we assume that the trajectories have been generated by applying a decision function D on the probability distribution over the commands, and that this function is linear, we do not need to specify $p(U_t|I_t, \tau_t)$. Indeed, distributing D in Equation 9 yields:

$$\begin{aligned} D(p(U_t|Z_{1:t-1})) &= \sum_{I_t,\tau_t} D(p(U_t|I_t,\tau_t))p(I_t,\tau_t|Z_{1:t-1}) \\ &= \sum_{I_t,\tau_t} \upsilon^{I_t}_{\tau_t} p(I_t,\tau_t|Z_{1:t-1}) \end{aligned} \quad (10)$$

where we can assume $D(p(U_t|I_t = i, \tau_t = j)) = \upsilon^i_j$. In the end, the command is a linear combination of commands from the training trajectories.

3.3.3 Taking sensor data into account.

The internal state is then finally updated using the observation:

$$p(I_t, \tau_t|Z_{1:t}) \propto p(Z_t|I_t, \tau_t)p(I_t, \tau_t|Z_{1:t-1}) \quad (11)$$

3.4 Complexity and parallelisability

Considering $L' = \max_i L_i$ and W the length of the update window used in the temporal transition model, the complexity of the algorithm is the following:

– Prediction update: $\mathcal{O}(N \times L' \times W)$, as for every trajectory and time indices, we have to sum over W samples and then multiply by a factor depending on the sum of the probability in that trajectory, that can be computed previously by summing for every trajectory over all time indices.
– Getting command: $\mathcal{O}(N \times L')$, as we have to sum over all trajectory and time indices.
– Taking sensor data into account: $\mathcal{O}(N \times L' \times k)$, as we have to sum over all trajectory and time indices as well as sensor dimensions.

As these steps are applied in sequence, the complexity is dominated by the heaviest. The complexity of our algorithm is thus $\mathcal{O}(N \times L' \times \max(W, k))$. Both W and k are fixed with respect to the number and length of trajectories. In our experiments, with $C_r = 3$, we have W ranging from 4 to 19 as θ_τ ranges from 0.01 to 1 and $k < 10$. Therefore, from a scalability perspective, the complexity is $\mathcal{O}(N \times L')$.

The operations for every trajectory (which typically have a similar length) can be conducted in parallel, with only the transition model and renormalisation moving probability mass between trajectories. These can be done at the same time and therefore require a synchronisation only once per time step. Therefore, this algorithm is well suited for parallel implementation on the CPU or the GPU.

3.5 Implementation

We have implemented our algorithm as an open-source Python module (BSD license)[1]. For performance reasons, the time-critical parts of the algorithm have been implemented in Cython, and optimised by looking at the generated C code. This provided a smooth path from the first experimental code to an efficient real-time implementation.

As our algorithm deals with probability distributions, these can quickly become infinitesimally small, leading to numerical problems. To alleviate these, if α is 0, our implementation adds a small value $C_\alpha = 10^{-150}$ (since we use double-precision floating point numbers) to the observation probability, and does a lot of sanity checks. This made sure that the latent space never became degenerated in any of our experiments.

As summary, in all our experiments, we used the following implementation constants:

– $C_s = 10$ Hz the frequency of record and replay.
– $C_r = 3$ the over-sampling resolution of $p(\tau)$.
– $C_w = 0.02$ the ratio of the mode of the $p(\tau_t|\tau_{t-1})$ sliding window at which the window is cut.
– $C_f = 10$ the count of last time steps to consider for stopping the replay.
– $C_{fp} = 0.9$ the ratio of probability mass that must be in last C_f time steps for the replay to stop.
– $C_\alpha = 10^{-150}$ constant to avoid numerical problems.

4 Capability analysis

As explained in Section 1, there are several distinct capabilities that our algorithm aims at capturing. Robustness to noise is handled by the Bayesian framework we use. Thus, we focus here on the other capabilities and their sensitivity with respect to the different meta parameters of our algorithm. Some of the capabilities have different, and even conflicting, optimal values for these meta parameters.

4.1 Experimental protocol

We use a small wheeled robot performing specific tasks in a virtual maze, that highlight the various capabilities of our algorithm. In this section, we explore these tasks in simulation. Section 6 will show the deployment of our algorithm on two different real robots.

The simulation is built using the Enki 2D simulator (Magnenat et al. 2009). The robot is a simulated e-puck robot (Mondada et al. 2009). It is a small (7.4 cm

[1] `https://github.com/bayesian-trajectory-replay`

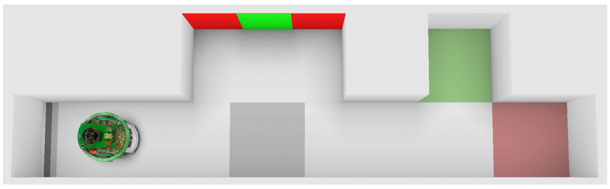


Fig. 2 Experimental setup used for analysing the capabilities of our algorithm. In this example, the wall of the wider area has a green poster surrounded by a red background.

diameter) differential-drive robot controlled by setting the speed of the left and right wheels. Eight infrared-based proximeters are mounted in a ring around the robot and a front-facing camera is used to distinguish the colour of the wall ahead of the robot. For this experiment, we use a single pixel in the centre of the camera and the 6 front- and side-facing proximeters. These proximeters have a highly non-linear response and Enki models them accordingly. In particular, their output saturates at distances outside the 1–5 cm range.

Figure 2 presents the environment: a corridor maze with a wider area in the middle and two rooms at the end. The ground of the left room is green and the ground of the straight room is red. The side of the wider area can be set to have a red or green colour, called poster, surrounded by a red or green background. Several tasks are available in this environment and are described in the following sub-sections.

For each task, we recorded several trajectories. The robot was placed at the origin with uniform noise of ± 0.5 cm on its position and $\pm 5°$ on its orientation. The sensor parameters were set to $\sigma_{\text{camera}} = 10\,\%$ and $\sigma_{\text{proximeter}} = 0.5$ cm. The demonstrations were performed by two different people (the two authors) by driving the robot with a joypad using visual feedback from a real-time rendering as show in Figure 2.

4.2 Complex task

We propose a complex task, the successful execution of which requires the different capabilities our algorithm provides. Starting from the left corridor, the task is to go in the wider area, turn 90° and check the colour of the poster for about 2 s, and then to go in the room of the matching colour. Both the colour of the central poster and of the surrounded wall can be independently set to either green or red. The robot is placed in the left corridor, facing right at three different starting positions: 6, 12 and 18 cm from the left wall. We recorded 120 training trajectories: 10 for each configuration of starting position, background colour and poster colour.

Figure 3 shows the traces of 120 replay trajectories, corresponding to 10 runs for each configuration as in

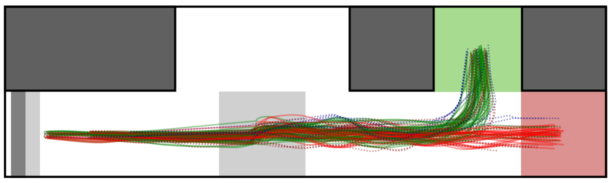


Fig. 3 Traces of 120 replay trajectories for the complex task, with parameters $\theta_\tau = 0.3$, $\theta_I = 10^{-8}$, and $\alpha = 0.01$. The colour indicates the target room. Full lines are successful runs, dashed lines are failed runs. Blue lines are runs with collisions.

the training. The full green and red traces correspond to successful runs to the respective rooms. The dashed ones correspond to entering the wrong room or stopping in the corridor. The blue traces correspond to collisions with the environment. Globally, the success rate is 61 %; the rate of collision is 11 % and the rate of going to the wrong room or stopping in the corridor is 28 %. The algorithm parameters are $\theta_\tau = 0.3$, $\theta_I = 10^{-8}$, and $\alpha = 0.01$. These values are used by default in the next sections. Section 4.6 analyses the search for these parameters.

4.3 Time-driven action: the look task

Being a trajectory tracker, our algorithm is able to take action as a function of time. The first part of the complex task is to get out of the left corridor and turn to look at the central wall colour. We call this the *look* task. The robot can achieve this task by moving forward in the corridor for some time until it is approximately at the level of the central poster and then by turning left in place by approximately 90°. If the robot is not in the dark grey zone (see Figure 2) or if the camera is not pointing at the poster once the turn is completed, the task is a failure. Note that the robot cannot deduce when to turn left from its current sensor measurements, but rather must count time since the wall on its left disappeared from the view of its proximeters.

We recorded 10 training trajectories and used our algorithm to execute the task. We run 500 replay trials. In most runs (88 %), the robot ends up looking at the correct part of the wall. Figure 4 shows the traces of a sample of 20 replays. Two of those are failures: in the first one, the robot turns too much and came to look backward; in the second one, the robot advances too much and does not turn back enough to compensate. The variability between the trajectories is due to the difference in starting pose, as well as noise on the sensors and actuators.

To study the effect of parameter of θ_τ, we vary, at replay, the starting position of the robot in the range of ± 6 cm, by steps of 1 cm, while using the same 10

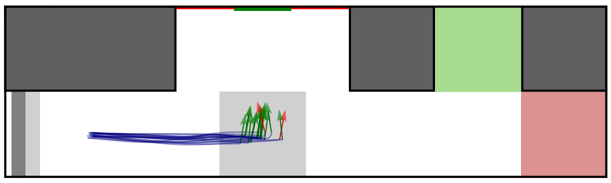


Fig. 4 Trace of 20 replay trajectories in the look task. Arrows indicate the final pose of the robot. The colour indicates whether the task is a success (green), or a failure (red).

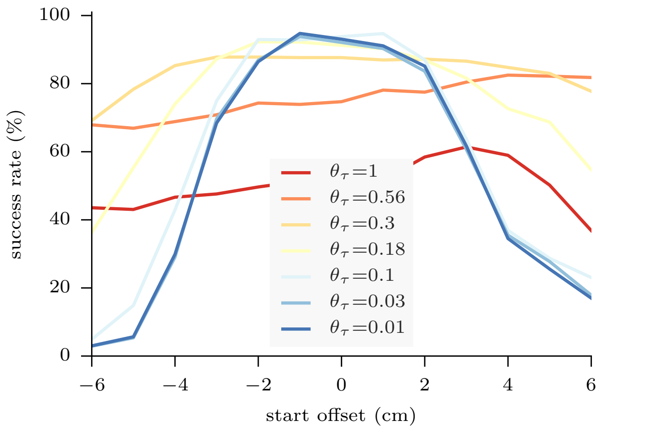


Fig. 5 Success rate for the look task as a function of starting offset for seven different values of θ_τ. The rate is computed based on 500 trials using all 10 training trajectories. $N = 10$, $\theta_I = 10^{-8}$, $\alpha = 0.01$

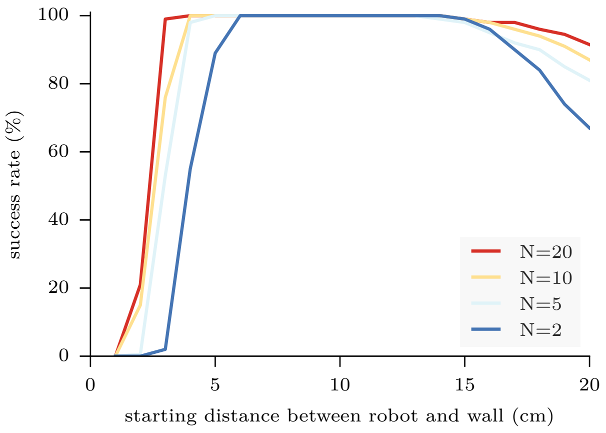


Fig. 6 Success rate for the dead-end task as a function of starting position for four different numbers of training trajectories. Each line shows the median rate of 500 random selections of N training trajectories out of 140. The rate is computed based on 100 trials. $\theta_\tau = 0.3$, $\theta_I = 10^{-8}$, $\alpha = 0.01$

training trajectories. Figure 5 shows that, as expected, for smaller offsets, the success rate is globally higher. However, we can see that bigger values of θ_τ lower the success rate while accepting a bigger start offset. Indeed, this parameter can be interpreted as the time accuracy of the replay with respect to the demonstrated trajectories. Therefore, if the robot is too strict in adhering to the time of the demonstrated trajectories, its ability to adapt to trajectories that should be longer or shorter due to a difference in the starting position is reduced.

4.4 Sensor-driven action: the dead-end task

Being a trajectory tracker, our algorithm does not simply follow a fixed trajectory but is also able to adapt to the sensory information and perform sensor-driven actions to a certain extent. This reactive capability can be necessary in the complex task, for instance to prevent the robot from colliding with walls. However, in order to study sensor-driven action in isolation, we designed a different task called the *dead-end* task.

In this task, the robot is placed facing left in the middle of the left corridor (see Figure 2) and must drive forward until it is at a distance of 1.5 cm from the wall (with a tolerance of 1 cm). The robot must not touch any walls, and thus must use its side proximeters to regulate its distance to the left and right walls, while using its front proximeter to decide when to stop. This task would be easy to formulate in a reactive form as the proximeters give all the necessary information.

We recorded 10 training trajectories for each of 14 different starting distances from the wall, from 6.3 to 20.3 cm by steps of 1 cm (corresponding to the robot's centre being from 10 to 24 cm from the wall). We test the algorithm by placing the robot at different starting distances from the wall, varying from 1 to 20 cm. This ensures that the test data do not exactly correspond to the training data.

We first want to assess if the robot is able to solve this task out of a limited set of training trajectories. For a given number N, we uniformly sample N trajectories out of our 140 training runs and compute the average success rate of 100 test runs with the robot starting at a given distance from the wall. Figure 6 shows the median success rate out of 500 samples of N trajectories, to account for variability of the results depending on the choice of training runs. We can see that for intermediate distances from 5 to 15 cm, in most cases the algorithm will succeed even with only 2 training trajectories. A greater number of trajectories is able to help our algorithm significantly extend the range of success.

In addition to the median, Figure 7 shows the 10 % and 90 % percentiles of the success rate. We can see that even though the median success rate is at 100 % for 2 training trajectories, at intermediate starting distances, the 10 % percentile can drop as low as 20 %. This is due to the need of diversity in the training trajectories for good generalisation, as the specific choice of training trajectories is important for the replay performance. The bigger the number of training trajectories, the lower

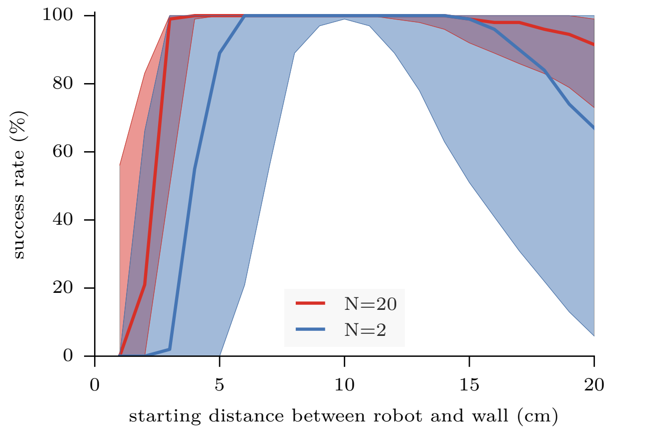


Fig. 7 Success rate for the dead-end task as a function of starting position for two different numbers of training trajectories. Each bold line shows the median rate of 500 random selections of N training trajectories out of 140. The coloured areas show the 10–90-quantiles. The rate is computed based on 100 trials. $\theta_\tau = 0.3$, $\theta_I = 10^{-8}$, $\alpha = 0.01$

the risk of lacking diversity, ensuring a more consistent performance.

An additional observation on Figure 7 is that the decrease in the success rate is not symmetric with respect to the reference position (12 cm). The decrease is slower but comes earlier when starting further away compared to starting closer to the wall. Indeed, the sensor data only give information about the wall at small distances and therefore do not give any information on the progress of the task up to that point. On the contrary, when the wall is observed earlier than expected, the mode of the probability distribution can be pushed forward in time due to the time stretching governed by θ_τ.

Figure 8 shows the median of the success rate for various values of θ_τ with a training set of 5 runs. It is quite clear that bigger values improve the overall performance.

Furthermore, Figure 9 shows that not only is the performance globally better with a bigger value of θ_τ but also that the performance becomes less dependent of the specific choice of training trajectories. This means that with a bigger value of θ_τ it is easier to interpolate between training data, and, as a consequence, that the performance is improved more this way than by increasing the number of training trajectories.

As a conclusion, θ_τ governs a trade-off between the precision of replay of a trajectory based on time and adaptability to different experimental conditions.

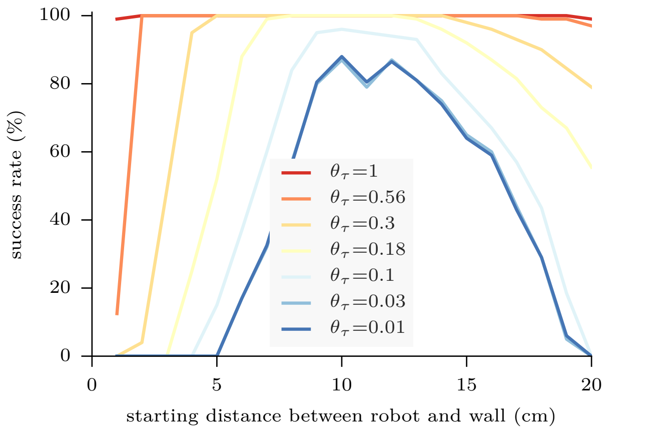


Fig. 8 Success rate for the dead-end task as a function of starting position for seven different values of θ_τ. Each line shows the median rate of 500 random selections of 5 training trajectories out of 140. The rate is computed based on 100 trials. $N = 5$, $\theta_I = 10^{-8}$, $\alpha = 0.01$

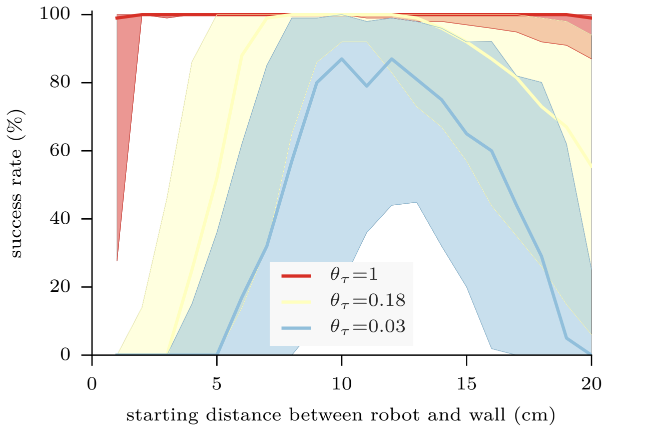


Fig. 9 Success rate for the dead-end task as a function of starting position for three different values of θ_τ. Each bold line shows the median rate of 500 random selections of 5 training trajectories out of 140. The rate is computed based on 100 trials. The coloured areas show the 10–90-quantiles. $N = 5$, $\theta_I = 10^{-8}$, $\alpha = 0.01$

4.5 Memory-driven action: the go-to-room task

Our algorithm is also able to choose actions based on what happened previously during the run. To test this capability, we designed a task, *go-to-room*, in which the robot should go in the room of the same colour as the central poster (green in the configuration of Figure 2). Both the colour of the central poster and of the surrounded wall can be independently set to either green or red. The challenge of the *go-to-room* task is that the signal to decide whether to turn left to the green room or continue straight into the red room is not present when the decision must be taken but, instead, significantly earlier when seeing the poster. We recorded 120 training trajectories (30 for each colour configuration) starting with the robot facing the central poster.

Figure 10 shows the outcome of 4000 replay trajectories as a function of the θ_I parameter. This shows that even if the training data is composed of trajectories

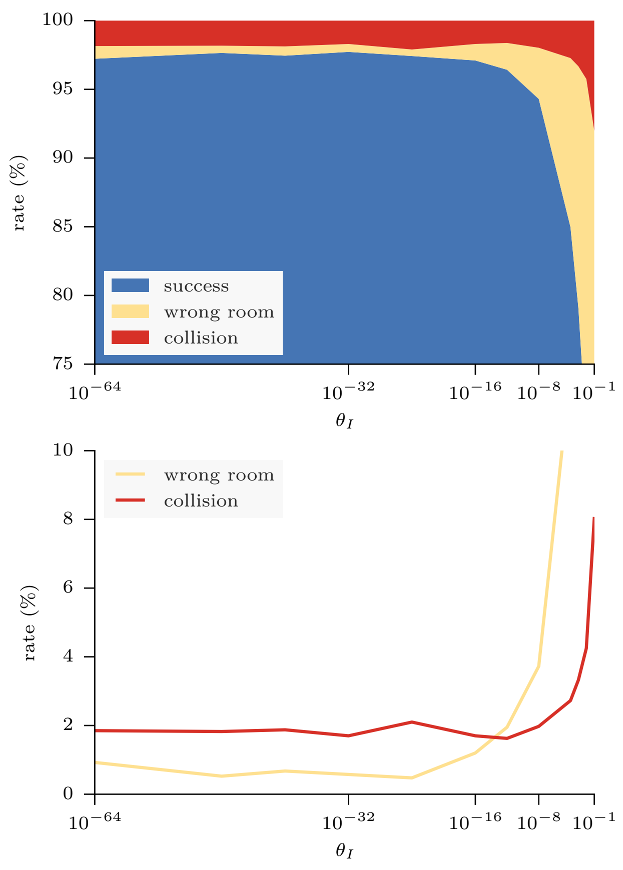


Fig. 10 Outcome of 4000 replay runs for the go-to-room task for each value of θ_I. "success" means the robot went to the designated room. "collision" means the robot collided with the wall. "wrong room" means the robot went to the other room. $\theta_\tau = 0.3$, $\alpha = 0.01$. Top: cumulated probability; bottom: individual probabilities of error conditions. Note that the ranges of the two plots are different.

going to either rooms in equal proportion, our algorithm is able to choose the correct room more often than by chance. This is done during the observation of the poster by the general lowering of the probabilities associated to the trajectories of the wrong colour.

We can see that with increasing values of θ_I, the chance to end in the wrong room – that is to have forgotten which room to go to – increases. At the same time, the collision rate decreases. This is due to the ability of our model to use experience from different trajectories to avoid obstacles. A choice of optimal value is difficult as the performance does not significantly depend on θ_I if it is not too large.

As a conclusion, θ_I governs a trade-off between the memorisation capability – tied to selected trajectories – and the adaptability – for which leveraging more trajectories is useful.

4.6 Optimal parameters for complex tasks

The analysis of the previous sections shows that each parameter influences the capabilities of the algorithm to reproduce a task with a need for either time-driven, sensor-driven or memory-driven actions. However, complex tasks will usually require all three capabilities and therefore these parameters must be chosen accordingly.

The complex task described in Section 4.2 can be seen as the combination of the *look* task (time-driven), the *go-to-room* task (memory-driven), and the *dead-end* task (sensor-driven).

4.6.1 Influence of the meta parameters.

Figure 11 shows the influence of each of the meta parameters on the success rate of the complex task. The values $\theta_\tau = 0.3$, $\theta_I = 10^{-8}$, $\alpha = 0.01$, shown as dashed lines, were chosen based on a coarse search.

The first parameter with significant influence is θ_τ, which was expected after the findings for the individual sub-tasks. Also, too high values for θ_I induce a large probability to go to the wrong room and to collide with the walls. This is similar to the results depicted in Figure 10.

However, there is now a new effect with lower values of θ_I: we observe that, despite a similar success rate, the collision rate increases. This illustrates the trade-off between leveraging trajectories with a history of non-matching observations in order to prevent collision and keeping a longer memory of past observations in order to guide present decision. The selected value of θ_I is therefore chosen so as to lower the chance to hit the wall while not significantly degrading the performance.

The last parameter, α, does not seem to have too much influence within this experiment. This is due to the actual simulation of the sensors which reproduces the saturation of the infra-red proximeters in a narrow range of distances (between 1 and 5 cm) with a corresponding noise parameter $\sigma = 0.5$ in our algorithm. Therefore, given that σ is 10 % of the sensor range, the α parameter, which prevents outlier observations from having disproportionate effects, is not really needed and could even be set to 0.

As can be seen in Figure 12, the success rate does not depend on the starting position. However, the colour configuration plays an important role, and can explain the variability in success rates. As expected, one can observe that the conflict situations (different colours for the wall and the poster) lead to a lower success rate. However, that is not only due to more confusion, but also to more collisions. A possible explanation is that, in conflict situations, the probability contrast between

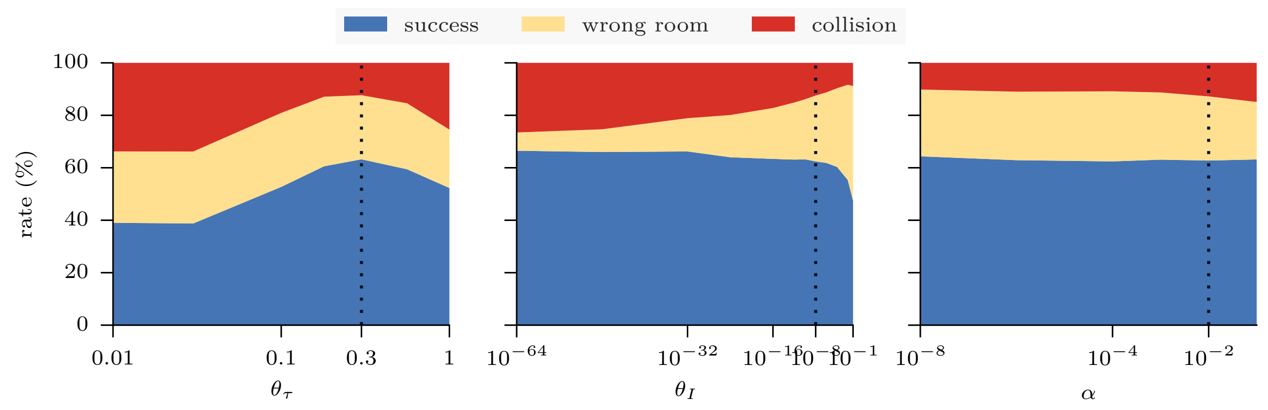


Fig. 11 Influence of the meta parameters on the outcome of the complex task. Distribution of the outcome as a function of the parameter values, computed based on 200 trials in each configuration (13,600 in total) using 120 training trajectories. The dashed lines show the selected parameter values.

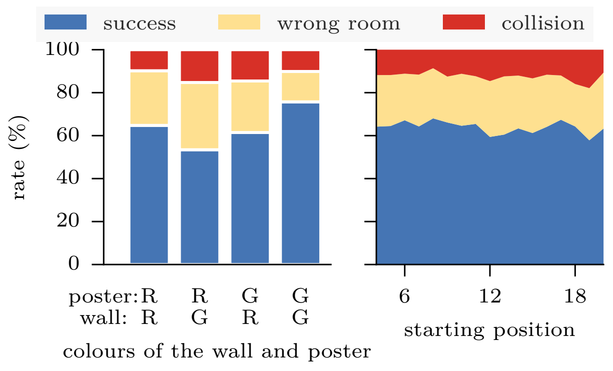


Fig. 12 Distribution of the outcomes based on the task configuration. Left: distributions according to the colours of the poster and the wall, based on 3,400 trials using 120 training trajectories. Right: distributions as a function of the starting position, based on 800 trials using 120 training trajectories.

correct and incorrect trajectories is lower, which leads the command to interpolate more and therefore stray further away from the demonstrated trajectories. In this case, the robot will have more difficulty to recover.

Furthermore, we observe that the robot has more difficulty to go to the red room than to go to the green room. This result seems counter-intuitive at first as going to the red room only requires to go straight whereas going to the green room requires turning left, the right amount at the right location. This can be explained by observing that, in going to the red room, the robot tends to stop prematurely. This is due to the robot having slightly lower acceleration during replay than recording (see Section 7). This problem also happens when the robot goes to the green room. However, in the green case, the corner is a salient observation helping the robot to keep track of the trajectory whereas in the red case, the robot only sees the right wall which is less informative.

4.6.2 Influence of the number of demonstrations.

A last element that can be studied is the influence of the number of demonstrations on the performance. Figure 13 shows the distributions of outcomes for 12, 36, or all 120 training trajectories. On the left, the trajectories are chosen evenly among the 12 different training conditions (3 starting positions, 2 wall colours and 2 poster colours) whereas on the right the trajectories are chosen randomly among the whole set of 120 recorded training trajectories. As expected, the success rate increases with a higher number of training trajectories. This is due to two factors: the collision rate decreases as there are more examples, and the relative rate of going to the wrong room also decreases.

With 12 trajectories evenly chosen, the system achieves a 53 % success rate with only one training trajectory per condition. When choosing the trajectories at random for 12 trajectories, there is no guarantee that there will be a training trajectory for each training case. We can observe on the right of Figure 13 that the overall success rate decreases to 49 %. The performance for 36 trajectories does not change significantly between the two selection strategies, presumably because the probability of not having an instance of each training configuration is small.

4.6.3 Computational time.

We report computational times running single-threaded on an Intel Core i9-8950HK processor in VirtualBox on a laptop computer. Using 12 training trajectories on the full task, each iteration took an average of 4 ms. For 120 trajectories, each iteration took an average of 45 ms. This ratio is consistent with the complexity analysis of Section 3.4. As our control loop runs at 10 Hz, the

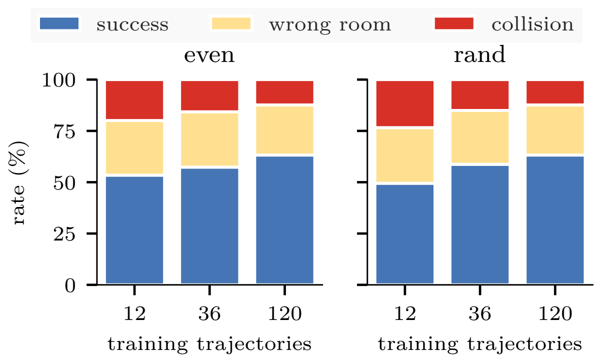


Fig. 13 Distribution of the outcomes based on the number of training trajectories, computed by averaging 68,000 trials using 20 sampling of trajectories (except for 120). Left: the training trajectories are distributed evenly among the 12 training conditions. Right: the training trajectories are chosen randomly among the 120 training trajectories.

performance is not an issue even with this large a number of training trajectories.

The average iteration time for 120 training trajectories can be broken down into 1 ms for the prediction step, 2 ms for the command step, and 42 ms for the observation step. This shows that the observation step, involving the complex observation model described in Section 3.2.2, is the major contributor to computational time.

As a conclusion, our system can achieve satisfactory performance on a complex task even with a low number of training examples, while keeping a reasonable computational load.

5 Comparison with Gaussian processes

To compare our method with existing approaches, we create a set of alternative Gaussian Process (GP) models that differ in how they consider time. We choose GP as a baseline because they form the core of several related works and have an elegant theoretical formulation (Rasmussen and Williams 2006). A GP is a stochastic process, that is a collection of variables, such that any finite collection of those variables follows a multivariate Gaussian distribution.

As such, it can be seen as a probability distribution over functions and, among others, it has been widely used for regression (Williams and Rasmussen 1996). In robotics, it has been the basis for some learning by demonstration approaches in which, a GP or a mixture of GP was learned to encode the sensori-motor trajectory (Calinon et al. 2010; Calinon and Lee 2016).

5.1 Models

Direct. The DGP model builds a single GP regressor from the time to the actuator commands, combining all considered training trajectories and ignoring the sensors. The function to learn can be written as:

$$t \mapsto U \tag{12}$$

Reactive. The RaGP model builds a single GP regressor from the sensor data to the actuator commands, combining all considered training trajectories and time steps. The function to learn is thus written as:

$$Z \mapsto U \tag{13}$$

Replay. The RpGP model builds, for each recorded time step, a GP regressor from the sensor data to the actuator commands, combining all considered training trajectories. The function to learn is therefore:

$$\forall t : Z_t \mapsto U_t \tag{14}$$

Implementation. All models are implemented in Python using *scikit-learn* (Pedregosa et al. 2011) and trained with the L-BFGS-B algorithm.

5.2 Comparison results

We can compare the performance of these models for each task.

5.2.1 Time-driven action

The *look* task consists in moving forward and turning on the spot in order to look at the poster. It is referred to as a time-driven action since, for a given starting position, executing commands with the same timing (open-loop control) is theoretically enough. In practice, such open-loop replay is limited as there is always noise in the motors and starting position and physical interactions are always slightly different.

The left side of Figure 14 shows the outcome distribution when the robot starts in the middle of the corridor and the models are trained from the same starting position. Besides success and collision, it can be useful to distinguish two other failure conditions: “not in front” means that the robot stopped either too early or too late and is not in the gray square depicted in Figure 2, whereas “not looking” means the robot camera is not pointing at the poster.

We can observe that, as expected, DGP has a high rate of success. Similarly, we can see that, as expected,

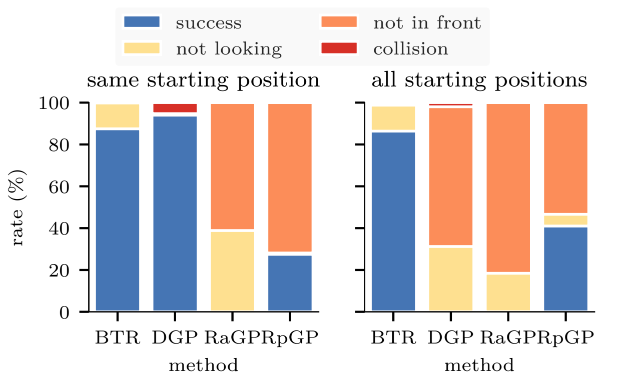


Fig. 14 Distribution of the outcomes on the *look* task for the different models. On the left, computed on 800 trials starting in the middle using, as training data, 40 trajectories with the robot starting in the middle. On the right, computed on 13,600 trials using 120 training trajectories with three different starting positions.

RaGP has trouble to stop at the right place and, when it does, to turn by the right amount.

What is less expected is the low probability of success of RpGP, which, most often, does not reach the proper place. This is due to the construction of the Gaussian Process: outside known sensor values, its mean is zero. Hence, when it observes something different than what it was trained for, the commands will be reduced until the robot stops. That is what happens here when, due to sensor uncertainties, the robot slightly slows down in the corridor. Thus, it reaches the exit later than in the recorded trajectories, leading to a large mismatch between these and its sensor observations, making it decelerate and quickly stop. This demonstrates the lack of robustness of RpGP.

The situation is different when the models are trained and tested with various starting positions (right side of Figure 14). Contrary to the case with a single starting position, DGP is not able to correctly perform the task. Indeed, all trajectories are mixed together and it starts turning while still moving forward, which makes it deviate from the central line of the corridor. It also has trouble turning by the right amount.

However, RpGP performs better since having more trajectories allows for more variability in sensor input that can help compensate for larger differences in various test conditions. The right of Figure 14 aggregates tests with starting positions sampled from a wider interval than the training data. For comparison, the performance of RpGP starting in the middle but with all the training trajectories, reaches 62.5 %. It is significantly higher than with only the trajectories starting in the middle (as can be seen on the left of Figure 14) but it is still below our model, which performs the best.

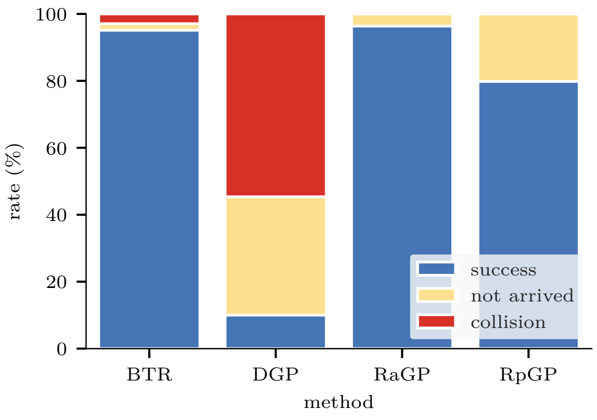


Fig. 15 Distribution of the outcomes on the *dead-end* task for the different models computed on 16,000 trials using 140 training trajectories.

5.2.2 Sensor-driven action

The *dead-end* task is purely reactive in that the robot needs to stop at a given distance from the wall while starting at different distances and with slightly different orientations.

Figure 15 shows the outcome of the various models for this task. As expected, RaGP performs nearly optimally. Actually, the cases where it fails correspond to the controller stopping too early. This only happens with the furthest test distances due to the termination condition of the controller arbitrarily chosen to be after 10 % more time steps than the longest duration training trajectories. A laxer termination condition would probably yield a perfect behavior.

The poor performance of DGP is also expected as it only mixes training commands regardless of sensor inputs. It only succeeds for starting distances close to the average of those of the training trajectories.

On the contrary, Figure 15 shows that RpGP can perform well enough when distances are not too large. When they are, at the end of the test run the sensors are still too far to perceive the wall while they do in all training trajectories. When this perception difference is too large, the zero-mean prior of the Gaussian process makes the robot stop.

On this task, our model shows a small probability of collision when starting very close to the wall. This makes it slightly worse for this reactive task than RaGP, the purely reactive model.

5.2.3 Memory-driven action

As explained above, in the *go-to-room* task, the robot needs to decide to go straight into the red room or turn left into the green room based on the colour of the poster.

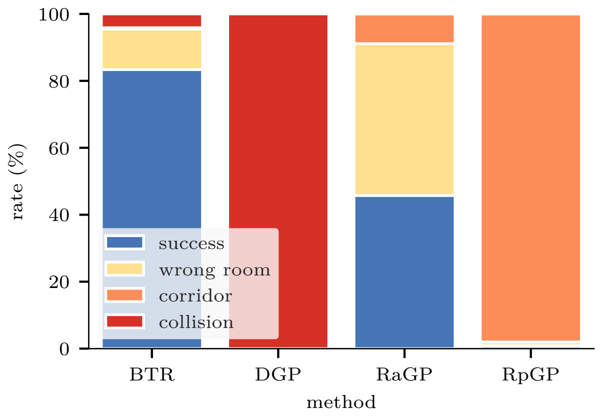


Fig. 16 Distribution of the outcomes on the *go-to-room* task for the different models computed on 800 trials using 120 training trajectories.

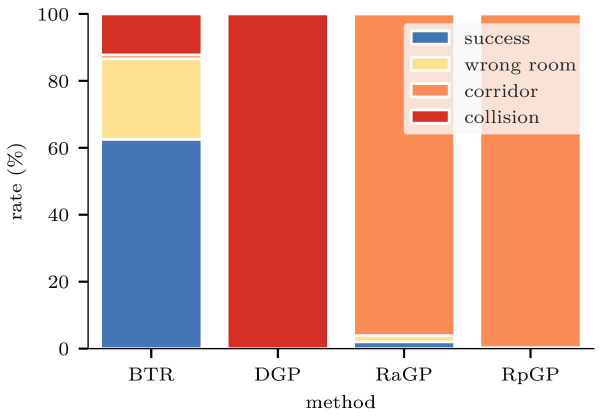


Fig. 17 Distribution of the outcomes on the complex task for the different models computed on 13,600 trials using 120 trajectories.

This cue is only observed at the start and is not present anymore at the actual time the decision must be taken.

As no GP model explicitly extracts this information, it is expected they all fail, as shown in Figure 16. The failure modes are different. For instance, as DGP essentially mixes the commands, it mostly hits the corner between the green and the red rooms.

RaGP exhibits a saner behaviour as it never hits the walls and mostly enters a room at random. In addition, around 10 % of the times, it stops before entering either room. This is again due to the zero-mean prior on the commands: when the sensor values are too different from the training data, the robot stops. This happens when the controller hesitates while choosing which room to go to and goes in between.

Finally, Figure 16 shows that RpGP never reaches a room but always stops before. This is the same phenomenon due to the zero-mean prior of stopping when seeing something too different from training data. While this is similar in RaGP, RpGP only uses the sensor values from the training trajectories at the same time step, which makes it less resilient than RaGP.

Our model does not explicitly check a color signal but tries to act as recalled from trajectories with similar observation histories. An element of this observation history is the color of the wall, which is why our model is able to succeed in more than 80 % of the cases.

5.2.4 Complex task

The complex task is the concatenation of the *look* and *go-to-room* tasks. The expectation for the success rate is thus to be lower than either task separately, which can be observed in Figure 17.

Besides our BTR model, no model is able to perform the task with any reliability. Only RaGP is able to rarely reach one of the rooms and, as a consequence, to reach the correct room at random. DGP and RpGP both perform similarly as the *go-to-room* task, that is, they go into collision or stay in the corridor, respectively.

This shows that, in order to perform the complex task, models ought to have a variety of capacities: sensor-, time-, and memory-driven action. While our simple tracking model possesses those capacities, it is not the case of any of the GP models tested.

6 Deployment on real mobile robots

This section demonstrates the deployment of our algorithm on two different robots. The first one is a small research robot performing a complex multi-step task. The second one is a field robot entering a trailer after climbing up a ramp.

6.1 Grasping an object with marXbot

A preliminary version of our algorithm was deployed on a small research mobile robot grasping an object and reported in (Magnenat et al. 2012b). In this section, we briefly summarise these results.

6.1.1 Protocol.

This experiment consists of grasping a polystyrene cube with the marXbot mobile robot (Bonani et al. 2010), which is equipped with a magnetic gripper (Figure 18, top-left). This gripper, described by Rochat et al. (2010), is composed of a permanent magnet that rotates within magnetic flow guiding elements, creating a stable on/off magnetic switch with low energy consumption. The robot has 5 degrees of freedom: the left and right track

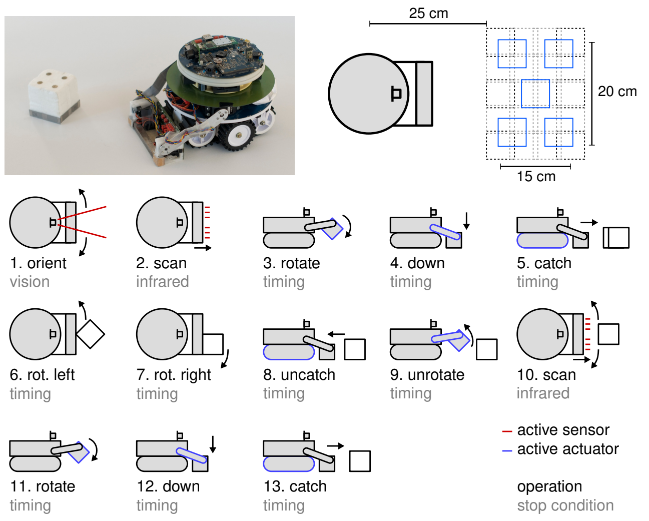


Fig. 18 The gripper-equipped marXbot robot with the cube (top-left), the experimental setup (top-right), and the sequence of operations to securely grasp a cube (bottom), from Magnenat et al. (2012a).

demonstrations	20							6
θ_I	10^{-4}	10^{-3}	$10^{-2.5}$	10^{-2}	$10^{-1.5}$	10^{-1}	$10^{-0.5}$	$10^{-0.5}$
success	9	11	7	13	13	13	14	11
timeout	5	2	6	0	0	1	1	3
grasp fail	0	2	1	1	1	0	0	1
exit	1	0	1	1	1	1	0	0

Table 1 Success rate of object grasping experiment for different values of θ_I based on 15 tests per value.

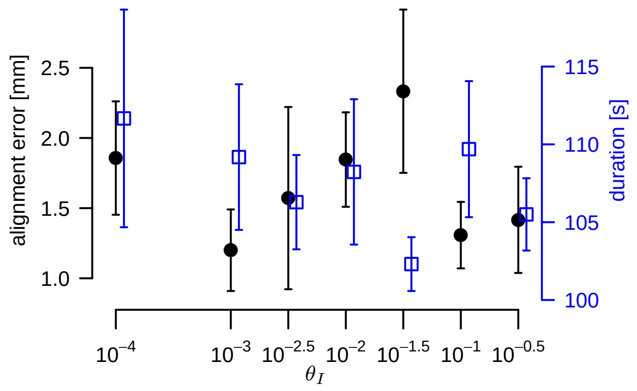


Fig. 19 The alignment error and run duration of object grasping experiment for different values of θ_I. Points are averages and bars show the standard deviation. These are computed on successful runs only. Note that the y-axes do not start at 0.

velocity, the elevation and tilt angle of the gripper, and the on/off position of its magnetic switch. The sensors consist of a camera and six infrared-based proximeters. The camera is pre-processed to return the position of the cube on the image's x-axis. The proximeters have a non-linear response with a good sensitivity between 1 and 3 cm, and almost saturate after 7 cm. They are pre-processed to return a linearised value.

The cube is placed at a distance of 25–40 cm from the robot, with a horizontal shift of ± 10 cm (Figure 18, top-right). The grasping procedure is complex: The robot must orient itself towards the cube, change the position of its gripper to scan the cube, advance until it is close enough, refine its orientation, turn its gripper back to the grasping position, fetch the cube, move sideways to ensure the cube is aligned, do another pass of scanning to align the cube in the centre of the gripper, and then finally fetch the cube again. Figure 18 (bottom) shows the exact sequence of operations with the description of the active sensors and actuators at a given step, and the condition to pass to the next step. This sequence was used as a guideline by the human driver and is given here purely as information for the reader, the algorithm was not aware of these steps in any way. We chose this problem because programming and tuning this behaviour has required a significant effort in a previous work reported by Magnenat et al. (2012a).

We use this scenario to validate the model on a safe physical system, to study the influence of the parameter θ_I, and to test the interpolation capabilities. We recorded 20 training trajectories with the cube at 20 different positions (dashed squares in Figure 18, top-right). Then, for 7 different values of θ_I, we tested 3 times the fetching of the cube at 5 different positions not present in the training data (solid blue squares in Figure 18, right). The other parameters are $\theta_\tau = 0.05$, $\alpha = 0$, $\sigma_{\text{cam}} = 8\,\%$, and $\sigma_{\text{proximeter}} = 0.5\,\text{cm}$.

When recording trajectories, a human driver tele-operated the robot with a joypad. The human was in another room than the robot, and was only given as input the same values as the algorithm, through a simplistic graphical user interface.

6.1.2 Results.

Table 1 shows the success rate. We sort failures into three categories: *timeout* indicates that the experiment duration exceeded 3 minutes without the robot completing the task, *grasp fail* indicates that the robot did not manage to grasp the cube or grasped it with an alignment error of more than 15 mm, and *exit* indicates that the robot exited the experiment area by mistake.

For a value of θ_I comprised between $10^{-0.5}$ and 10^{-2}, the success rate is high at about 85–90 %. For smaller values, it drops because, at some point, the robot either does not move any more or performs a movement repeatedly. When runs are successful, the alignment error is always small and the duration relatively constant

(Figure 19). We believe that a large θ_I leads to the best performance because our training trajectories essentially differ at the beginning, and therefore a large θ_I allows for more fine tuning of the behaviour afterwards, by jumping to a different trajectory that fits better to the observation.

Most of the failures (60 %) are linked to the robot stopping indefinitely, due to a fixed or cyclic distribution on I, τ. We attribute this effect to three causes. First, motor commands are discretised with a relatively low resolution, preventing the robot from moving if, for example, the motor command is 0.4. We could alleviate this problem through temporal dithering, by probabilistically selecting the nearest integers in proportion to their distance. The second problem is due to the servo motors that actuate the gripper. As they are controlled in position, they do not cope well with rapid changes of set points. We have observed that they perform the best when the battery is fully charged, because their speed is directly proportional to the battery voltage. Third, contrary to our current model, this early work used a simplified temporal transition model with no temporal sub-sampling:

$$p(\tau_t|\tau_{t-1}) = \begin{cases} \theta_\tau & \text{if } \tau_t = \tau_{t-1} \\ 1 - 2\theta_\tau & \text{if } \tau_t = \tau_{t-1} + 1 \\ \theta_\tau & \text{if } \tau_t = \tau_{t-1} + 2 \\ 0 & \text{otherwise} \end{cases} \qquad (15)$$

This allowed the probability distribution to stay “in place”, while Equation 5 forces it to “go forward”. The other types of failure are linked to the robot missing the cube or exiting the experimentation area.

To test the interpolation capabilities, we have used only 6 runs out of 20 (dashed black squares in Figure 18, right) and applied the same test procedure as before, with θ_I fixed to $10^{-0.5}$. Out of 15 test runs, 11 were successful, 3 failed because the robot stopped indefinitely, and 1 failed because the robot did not align properly with the cube. In the successful runs, the mean error was 2.1 mm and the mean duration was 99.5 s, which is similar to the results with 20 training runs. This test shows that our model is able to interpolate between training data and that its performances degrade gracefully when less data are available.

Overall, this experiment shows that our algorithm is able to reproduce a complex sequence of actions based on a limited set of demonstrations and to combine, at replay time, the beginning of a recorded trajectory with the end of another one.

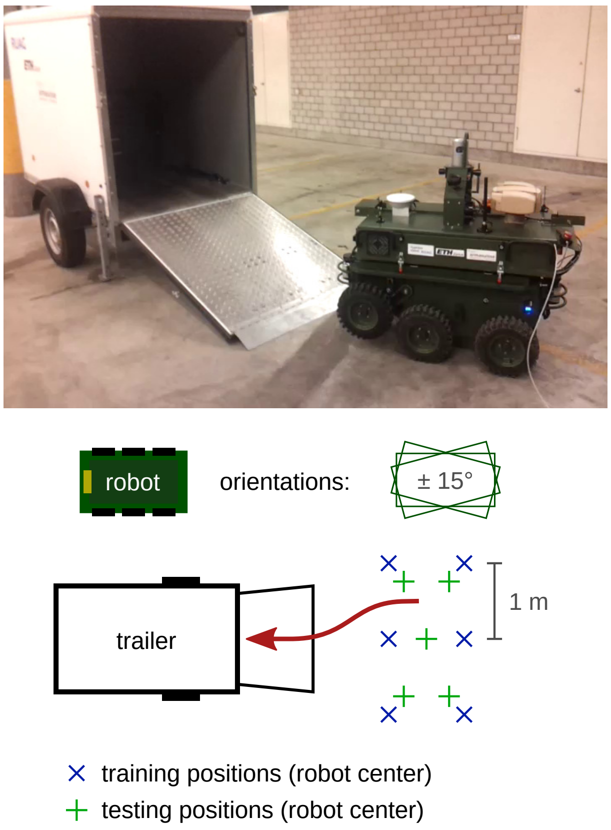

robot
orientations:
± 15°
trailer
1 m
× training positions (robot center)
+ testing positions (robot center)

Fig. 20 The artor robot (top) and the trailer entering experimental setup (bottom).

6.2 Entering a trailer with artor

After demonstrating our algorithm on a desktop research robot, we show that it can control a 200 kg robot (Figure 20, top) to do a common task (entering its trailer, as it can be seen in the video[2]). It achieves that with only a handful of demonstrations, which is much faster than programming the robot by hand.

6.2.1 Protocol.

We recorded 18 trajectories, with the robot in front of the trailer at 3 different parallel positions, 2 distances, and 3 orientations (Figure 20, bottom). Each recording lasts about 15 s and inputs/outputs are sampled at 10 Hz. The inputs are the roll and pitch angles from the IMU and 5 processed values from a front-mounted horizontal SICK lidar. These values are the mean of distances (clipped to 3 meters) of the following arcs covered by the lidar: [0°:60°[, [60°:80°[, [80°:100°],]100°:120°],]120°:180°]. The outputs are the velocities of the left and right wheels.

When recording trajectories, a human driver controlled the robot through a wireless joypad to enter the

[2] The video is available at `http://stephane.magnenat.net/videos/artor_entering_trailer.mp4`

	count	percentage
success	38	85 %
missed the ramp	6	13 %
failed to stop	1	2 %

Table 2 Results of trailer entering experiment

trailer, which is about 70 cm up from the ground. Under a close watch, the robot was first brought in front of the ramp, then aligned to the ramp, then driven straight until it fully entered the trailer, and finally stopped.

Note that, contrary to similar teleoperation tasks in the literature (Calinon 2016; Havoutis and Calinon 2019), we do not know the configuration of the robot nor the relative position of the trailer.

6.2.2 Results.

We set the parameters to the followings: $N = 18$, $\theta_\tau = 0.3$, $\theta_I = 10^{-8}$, $\alpha = 0.01$, $\sigma_{\text{IMU}} = 5$, $\sigma_{\text{lidar}} = 10\,\text{cm}$. We tested 5 different starting positions (front-left, front-right, centre-centre, back-left, back-right) within the area covered by the training data. For each position, we tested three different orientations (left, centred, right) and for each orientation we ran 3 trials. In total this corresponds to 45 trials.

As Table 2 shows, most of the trials were successful. In case of failure, the robot missed the ramp because it did not turn enough. This is due to the combined effect of the averaging of command outputs from all trajectories, and time-driven action. This led the robot to try to go to the ramp phase before completing the alignment phase. It would be interesting to explore whether choosing the output of the most likely trajectory, rather averaging over all trajectories, would lead to better results. We set the parameter α to 0.01 in order to bound the cost of a sensor reading being totally different than in the recorded trajectory. We believe that it is important in this experiment, because when the robot is aligning to the ramp, the distance perceived by its laser beams can change from 1 m to 20 m, with a $\sigma_{\text{lidar}} = 10\,\text{cm}$. Unfortunately, due to logistic constraints, we were not able to experimentally test a different output selection strategy or the effect of different values of α.

7 Discussion

The computational complexity of our algorithm is essentially the product of the number of trajectories and their average length. This is tractable on current laptops and embedded systems, for hundreds of trajectories and thousands of time steps, as demonstrated in Section 4.6.3. Should more data need to be processed, our equations are also well suited for parallelisation, using SIMD instructions or through a GPU implementation. Therefore, our method is not limited by the computational load.

As explained in Section 3.3.2, our model interpolates between the observed trajectories according to their estimated probabilities. A slight limitation of this approach, as opposed to taking the best command at each time step, is that, when uncertain, it can lead to middle-ground solutions between different modes. The consequence is that, during transient periods such as accelerations that may happen at slightly different times, the replay progresses slower with respect to the recordings. This might lead to failures later on if there are no observations able to realign the replay. This effect can be observed in our capability analysis when the robot has to go to the red room (see Section 4.6.1). An approach to limit this issue might be to track the sensorimotor state of the robot as a time window instead of just a single time step. For instance, Berlati et al. (2020) resolve ambiguity by predicting multiple trajectories with a recurrent neural network instead of just a single time step or a single trajectory.

Because our model takes little a-priori knowledge, its generalisation ability is limited. Given meaningful σ_k, it can interpolate but not properly extrapolate, in other words it cannot "guess" the appropriate action. It can, however, use parts of different trajectories during execution. We believe that in many practical scenarios, this is sufficient as extrapolation is not needed or desired. Note that the sensor space should be isotropic, otherwise σ_k has little meaning. However, despite this theoretical limitation, our algorithm worked well on linearised outputs of non-linear sensors.

In Section 4, we discussed that in order to take actions in function of sensory inputs, θ_τ should be large. Similarly, to be able to use parts of different trajectories, θ_I should also be large. However, to consider past observations to decide future actions and consider timing to trigger actions, they should both be small. These contradictory requirements could only be reconciled by creating a new layer of abstraction, in which symbols representing meaningful events and reusable trajectory blocks could be extracted from raw data as described in the context of mapping by Beeson et al. (2010). This is a difficult problem in general, so currently research has explored the following sub-problems:

– How to select a subset of variables that are relevant to control the current action of the robot. For example, Dong and Williams (2012) has studied how the target frame of the trajectory changes in the course of the task. Future research should explore how to generically identify those variables that sup-

port motor commands, as explored by Konidaris et al. (2012).
- How to build state machines from a set of demonstrations in specific contexts, as by Niekum et al. (2015). This field should be explored in a more principled way to find generic and reusable approaches. A key feature of state machines is the ability to handle tasks with loops. It would be interesting to explore whether our algorithm can handle loops by mixing the current transition model with a tiny bit of a uniform distribution. Nevertheless, we believe that in most cases this would dilute the commands too much, and therefore a symbolic approach to loop detection is probably more robust.
- How to non-linearly reduce the dimensionality of sensor values prior to using them as input for the algorithm. For example, deep neural networks such as auto-encoders have good properties as shown by Wang et al. (2016), and could be trained on all recorded trajectories. This would improve the run-time computational performance of the algorithm, as the observation would be of lower dimension. However, a key feature of our algorithm is that it is a white box, allowing an easy inspection and understanding of its behaviour.

In the light of these different research directions, our work provides a good basis upon which to explore future questions. Our meta parameters, while not trivially interpretable, have each explicit, direct and separate effects on the algorithm. Moreover, they are specified in the primary space of a white box algorithm. In that sense, they are easier to comprehend than meta parameters of more complex algorithms, for example Hierarchical Dirichlet Processes, which are factors related to a latent space rather than the operating space of the robot.

We believe that to be useful, future systems should be as generic and simple as possible and their studies should provide an understanding of the effect of their meta-parameters. In particular, in the spirit of Occam's razor, we believe that adding layers should be justified by a general need in a variety of applications and that these layers should be as simple and non-parametric as possible. A valuable future work would thus be to conduct an A/B testing and validate it through a user study.

8 Conclusion

In this paper, we presented a system, based on a non-parametric Bayesian model, that is able to perform multi-step tasks from demonstration. Our specific contributions are:

- On the theoretical side, we have explored how far we can go with an approach that makes minimal assumptions on the application scenario. We have shown that as long as the latent space is of low dimensionality with respect to the number of demonstrations, our approach works well and is relatively robust to changes in the value of the meta parameters.
- On the robotics side, we have demonstrated the system on a task of a research robot that proved hard to program by hand, and on a real task of a field robot.
- On the software side, we provide an open-source implementation for the research community and the industry to experiment with and deploy on their own robots.

Compared to related work, our system is easier to deploy because it has fewer meta parameters, while still providing good performance in the case of simple tele-operated robots. We believe that such a system can be of tremendous help for developing and deploying robotics applications. Indeed, the current requirement of choosing by hand between many different approaches and then exploring their many parameters is a major obstacle to the deployment of programming by demonstration, and more generally robotics technology.

Acknowledgements This work was mostly conducted at the Autonomous Systems Laboratory of ETH Zürich. We thank Philipp Krüsi for his help conducting the experiments, and Cédric Pradalier for his insightful comments on the algorithm. We thank the Mobots group of Francesco Mondada at EPFL for the access to the marXbot robot.

This work was supported by the NIFTi (FP7-247870), myCopter (FP7-266470) and Noptilus (FP7-270180) European projects and by Willow Garage through a 3-month visiting researcher grant.

References

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 1, 2004.

Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13):1608–1639, November 2010. doi: 10.1177/0278364910371999.

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 2009.

Patrick Beeson, Joseph Modayil, and Benjamin Kuipers. Factoring the Mapping Problem: Mobile Robot Map-building in the Hybrid Spatial Semantic Hierarchy. *The International Journal of Robotics Research*, 29(4):428 –459, April 2010. doi: 10.1177/0278364909100586.

A. Berlati, O. Scheel, L. D. Stefano, and F. Tombari. Ambiguity in sequential data: Predicting uncertain futures with recurrent models. *IEEE Robotics and Automation Letters*, 5(2):2935–2942, 2020. doi: 10.1109/LRA.2020.2974716.

Michael Bonani, Valentin Longchamp, Stéphane Magnenat, Philippe Rétornaz, Daniel Burnier, Gilles Roulet, Florian Vaussard, Hannes Bleuler, and Francesco Mondada. The marxbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In *Proc. of the IEEE/RSJ International Conference Intelligent Robots and Systems (IROS)*, pages 4187–4193. IEEE, 2010.

Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1): 1–29, 2016. doi: 10.1007/s11370-015-0187-9.

Sylvain Calinon and Dongheui Lee. Learning Control. In Ambarish Goswami and Prahlad Vadakkepat, editors, *Humanoid Robotics: A Reference*, pages 1–52. Springer Netherlands, 2016. doi: 10.1007/978-94-007-7194-9_68-1.

Sylvain Calinon, Florent Guenter, and Aude G. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):286–298, 2007. doi: 10.1109/TSMCB.2006.886952.

Sylvain Calinon, Florent D'Halluin, Eric L. Sauser, Darwin G. Caldwell, and Aude G. Billard. Learning and reproduction of gestures by imitation. *Robotics & Automation Magazine, IEEE*, 17(2):44–54, 2010. doi: 10.1109/MRA.2010.936947.

Sotirios P. Chatzis, Dimitrios Korkinof, and Yiannis Demiris. A nonparametric Bayesian approach toward robot learning by demonstration. *Robotics and Autonomous Systems*, 60 (6):789–802, June 2012. doi: 10.1016/j.robot.2012.02.005.

Nutan Chen, Maximilian Karl, and Patrick van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 629–636. IEEE, 2016.

Christian Daniel, Gerhard Neumann, and Jan Peters. Learning concurrent motor skills in versatile solution spaces. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3591–3597. IEEE, 2012.

J. DelPreto, J. I. Lipton, L. Sanneman, A. J. Fay, C. Fourie, C. Choi, and D. Rus. Helping robots learn: A human-robot master-apprentice model using demonstrations via virtual reality teleoperation. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 10226–10233, 2020. doi: 10.1109/ICRA40945.2020.9196754.

Shuonan Dong and Brian Williams. Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes. *International Journal of Social Robotics*, pages 1–12, 2012. doi: 10.1007/s12369-012-0155-x.

Paul Furgale and Timothy D. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, September 2010. doi: 10.1002/rob.20342.

Daniel H. Grollman and Odest Chadwicke Jenkins. Can we learn finite state machine robot controllers from interactive demonstration? In Olivier Sigaud and Jan Peters, editors, *From Motor Learning to Interaction Learning in Robots*, volume 264 of *Studies in Computational Intelligence*, pages 407–430. Springer, 2010. doi: 10.1007/978-3-642-05181-4_17.

Ioannis Havoutis and Sylvain Calinon. Learning from demonstration for semi-autonomous teleoperation. *Autonomous Robots*, 43:713–726, 2019.

Guillaume Infantes, Malik Ghallab, and Félix Ingrand. Learning the behavior model of a robot. *Autonomous Robots*, 30: 157–177, 2011. doi: 10.1007/s10514-010-9212-1.

Maximilian Karl, Maximilian Soelch, Philip Becker-Ehmck, Djamel Benbouzid, Patrick van der Smagt, and Justin Bayer. Deep variational Bayes filters: Unsupervised learning of state space models from raw data. In *5th International Conference on Learning Representations (ICLR)*, Toulon, 2017.

E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*. World Scientific, 2004.

S. M. Khansari-Zadeh and A. Billard. Learning stable non-linear dynamical systems with Gaussian Mixture Models. *IEEE Transactions on Robotics*, 27:943–957, 2011.

Nathan Koenig and Maja J. Matarić. Robot life-long task learning from human demonstrations: a Bayesian approach. *Autonomous Robots*, 41:1173–1188, 2017.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

J. Loula, K. Allen, T. Silver, and J. Tenenbaum. Learning constraint-based planning models from demonstrations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

Kirk MacTavish, Michael Paton, and Timothy D. Barfoot. Selective memory: Recalling relevant experience for long-term visual localization. *Journal of Field Robotics*, 35(8): 1265–1292, November 2018. doi: 10.1002/rob.21838.

Stéphane Magnenat, Markus Waibel, and Antoine Beyeler. Enki – an open source fast 2d robot simulator. https://github.com/enki-community/enki, 2009.

Stéphane Magnenat, Roland Philippsen, and Francesco Mondada. Autonomous construction using scarce resources in unknown environments. *Autonomous Robots*, 33:467–485, 2012a. doi: 10.1007/s10514-012-9296-x.

Stéphane Magnenat, Cédric Pradalier, and Francis Colas. Towards non-parametric bayesian learning of robot behaviors from demonstration. In *Bayesian Nonparametric Models For Reliable Planning And Decision-Making Under Uncertainty, NIPS 2012*, 2012b.

Jérome Maye, Rudolph Triebel, Luciano Spinello, and Roland Siegwart. Bayesian on-line learning of driving behaviors. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4341–4346. IEEE, 2011. doi: 10.1109/ICRA.2011.5980414.

F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proc. of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65, 2009.

Manuel Mühlig, Michael Gienger, and Jochen J. Steil. Interactive imitation learning of object movement skills. *Autonomous Robots*, 32(2):97–114, 2012. doi: 10.1007/s10514-011-9261-0.

Andrew Y Ng and Stuart J Russell. Algorithms for inverse reinforcement learning. In *ICML*, volume 1, page 2, 2000.

Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G. Barto. Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157, February 2015. doi: 10.1177/0278364914554471.

Peter Pastor, Mrinal Kalakrishnan, Franziska Meier, Freek Stulp, Jonas Buchli, Evangelos Theodorou, and Stefan Schaal. From dynamic movement primitives to associative skill memories. *Robotics and Autonomous Systems*, 61(4):

351–361, April 2013. doi: 10.1016/j.robot.2012.09.017.

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*, 12:2825–2830, 2011.

Cédric Pradalier and Pierre Bessière. Perceptual navigation around a sensori-motor trajectory. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3831–3836. IEEE, 2004.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

Frédéric Rochat, Patrick Schoeneich, Michael Bonani, Stéphane Magnenat, Francesco Mondada, Hannes Bleuler, and Christoph Hürzeler. Design of magnetic switchable device (MSD) and applications in climbing robot. In *Proc. of the 13th International Conference on Climbing and Walking Robots*, pages 375–382. World Scientific, 2010.

Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory - COLT' 98*, pages 101–103, 1998. doi: 10.1145/279943.279964.

J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33:1251–1270, 2014.

Bruno Siciliano and Oussama Khatib, editors. *Springer handbook of robotics.* Springer, 2016.

N. Soans, E. Asali, Y. Hong, and P. Doshi. Sa-net: Robust state-action recognition for learning from observations. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2153–2159, 2020. doi: 10.1109/ICRA40945.2020.9197393.

Dizan Vasquez, Billy Okal, and Kai Arras. Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1341 – 1346, 2013. doi: 10.1109/IROS.2014.6942731.

Yasi Wang, Hongxun Yao, and Sicgheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, April 2016. doi: 10.1016/j.neucom.2015.08.104.

Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for regression. In *Advances in neural information processing systems*, pages 514–520, 1996.

Xianghai Wu and Jonathan Kofman. Human-inspired robot task learning from human teaching. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3334–3339. IEEE, 2008. doi: 10.1109/ROBOT.2008.4543719.

L. Xie, A. Markham, and N. Trigoni. Snapnav: Learning mapless visual navigation with sparse directional guidance and visual reference. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1682–1688, 2020. doi: 10.1109/ICRA40945.2020.9197523.

W. Yang, N. Strokina, N. Serbenyuk, R. Ghabcheloo, and J. Kämäräinen. Learning a pile loading controller from demonstrations. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4427–4433, 2020. doi: 10.1109/ICRA40945.2020.9196907.