Online-Learning and Planning in High Dimensions with Finite Element Goal Babbling

Pontus Loviken
AI Lab, SoftBank Robotics Europe
Plymouth University, UK
Email: ploviken@softbankrobotics.com

Nikolas Hemion
AI Lab, SoftBank Robotics Europe
Email: nhemion@softbankrobotics.com

***Abstract*—Goal babbling (GB) has proved to be a powerful tool in online learning of inverse kinematic models of high-dimensional redundant robots that are acting in low dimensional sensor-spaces. To only look for inverse models is however not sufficient. An inverse model will only tell the robot *what* posture it should have in order to reach a goal, but not *how* to reach that posture. As many environments restrict what motions are possible this becomes a limitation.**

This paper introduces a new method, Finite Element Goal Babbling (FEGB), that presents a natural extension to GB. By partitioning the sensor-space into a disjoint set of finite elements where every element is seen as an independent GB problem, a planning module can be added by observing transitions between the different elements.

The method is evaluated on a high dimensional planar arm, acting in an environment that restricts its movements. The goal is to learn to control the position of the end-effector so that it can reach any position in the environment. The results show that FEGB is able to learn such control rapidly while naturally dealing with stationary obstacles and workspace limits that would prohibit the applicability of GB.

I. Introduction

Many behaviors can be naturally described in terms of movement of a given body part in a well-defined space: reaching means moving the hand towards the location of an object, standing up can be described as moving the head upwards, locomotion can be said to be the displacement of the body's center of mass in the direction of travel. In the robotics literature, these spaces are referred to as *operational* or *task space*, and are distinct from the *configuration space* or *motor space*, which the robot has control over.

Within developmental robotics [1] a natural question is how to allow a robot to gain control over a task space, given its control over a motor space. One of the most common approaches in the literature is to let the robot explore what it can do through "motor babbling", that is, moving some or all of its motors in a random way and observing what effects this has on its sensors. A major problem with this approach is that the motor spaces of all but the most simple robots are very high-dimensional. Consequently, it is theoretically impossible for the robot to exhaustively explore the entire motor space within a reasonable amount of time ("within a lifetime"), due to the so-called curse of dimensionality.

An alternative strategy, which circumvents this difficulty, is to let robots explore the task space instead of the motor space.

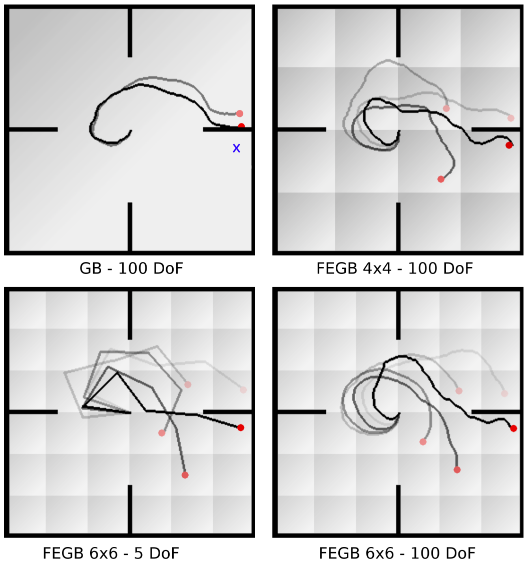

Fig. 1. By viewing different regions of the task space (here spanned by the position of the end effector) as discrete states of a Markov Decision Process (MDP), FEGB is able to plan trajectories around obstacles that would stop traditional goal babbling (GB) approaches where only an inverse model is learned. This figure shows the same attempted transition for four different settings after 10000 iterations of training. Notable is that GB can not reach the goal $\times$, as it is unable to plan a motion around the wall.

As a result, the robot efficiently learns *one way* (as opposed to learning all possible ways) of navigating the task space. One proposed method for achieving this is *goal babbling* [2], among others [3], [4], [5]. Goal babbling exploits the fact that many task spaces are naturally low-dimensional (for example, any translation of a body part can be described by a three-dimensional trajectory). This allows even robots with a high number of degrees of freedom to explore the entire task space within a reasonably short amount of time. A limitation of most previous implementations of goal babbling is however that it has mainly focused on settings where there exists a forward model that maps motor actions to specific outcomes in the task space. This applies to situations where the starting condition for every action is always the same, or where the probability for the agent to reach a certain motor state is independent of its current motor state or position in the task space. In reality such assumptions are clearly not always applicable. One such example can be seen in Figure 1 to the upper left. It is not sufficient for the agent to know what motor state to reach if it can not reach it, as in this case with a wall in the way.

This paper introduces Finite Element Goal Babbling (FEGB), an extension to the goal babbling framework. FEGB shows that by making some restrictions to the goal babbling framework, it is possible to combine it with a higher level *planner* that abstracts different regions of the task space as discrete states of a Markov Decision Process (MDP). By learning transitional probabilities between these different states in the MDP, it is then possible for the planner to identify sequences of regions an agent has to move through in order to reach a desired goal. The results shows that the addition of such a planning capability allows the agent to learn online how to efficiently reach goals even in environments where movements have to be planned, and without losing the ability of goal babbling to find such motor policies in a training time that is more or less untangled from the dimensionality of the motor space.
Code: https://github.com/loviken/fegb/tree/icdl-epirob-2017/
Videos: http://pontusloviken.com/icdl-epirob-2017/

II. Related Work

Goal babbling and related ideas have been shown to effectively permit learning across a number of domains, for example reaching [2], tool use [6], and speech production [7], [8]. The motor space varies in these cases from action spaces spanned by joint angle configurations (as in [2] or [9]) where a motor action is represented by a goal joint configuration, also referred to as a goal posture, to which the agent tries to move, to motor actions that represent extended motions in time such as Dynamic Movement Primitives (as in [6]) or Central Pattern Generators (as in parts of [10]).

The latter case is generally restricted to scenarios where the system returns to an initial state after each action. In the former case there are generally two different approaches depending on whether the goal is to just explore a task space, or to also allow an agent efficient control over the task space. If the goal is only exploration it is usually sufficient to perturb a previously seen posture that was close to a given goal [9]. This efficiently explores the task space but gives no information of what posture can be reached from any other posture, and might try to use vastly different postures for goals that are very close to each other. In cases where the motion between different postures has to be taken into account another approach can be employed. This approach focuses on actively finding one *consistent* continuous inverse function that allows the agent to move in a straight motion between any two points in the task space. Such mappings are usually found through the use of a "home posture", a relaxed posture that the agent frequently returns to, and a weighting scheme that enforces consistency in the inverse model. Examples of such approaches are found in [2], [10] and [11]. Beside limiting the inverse model to postures similar to the home posture, this approach also requires task spaces where such consistent models are possible, which is not the case if there exist discontinuities in

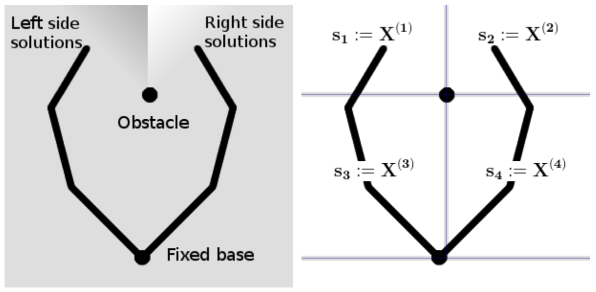
Left side solutions
Right side solutions
Obstacle
Fixed base
$\mathbf{s_1 := X^{(1)}}$
$\mathbf{s_2 := X^{(2)}}$
$\mathbf{s_3 := X^{(3)}}$
$\mathbf{s_4 := X^{(4)}}$

Fig. 2. *Left.* A planar robot arm that is fixed by the base with an obstacle within its workspace can not have a continuous mapping with one posture to every end effector position. Since the robot can reach past the obstacle from both sides there has to exist a discontinuity behind the obstacle where the agent decides whether to pass the obstacle on the right or the left side. *Right.* It is often possible to split a discontinuous workspace into smaller regions/states where the mapping within every region can be continuous, while dispatching discontinuities to the boundaries in-between. In this case the discontinuity is placed between $X^{(1)}$ and $X^{(2)}$, so that a transition between the regions must go through $X^{(3)}, X^{(4)}$.

the mapping from task- to motor-space. This is the case for example as soon as an object is introduced within its reachable space (see figure 2) or even if the agent is able to reach around its base.

For those interested in exploring traditional goal babbling frameworks is Explauto One [12] an easy to use platform.

Finite Element Goal Babbling is different from goal babbling in that it integrates the low-level control of the goal babbling framework with higher level planning. This is a trending topic in Machine Learning (see e.g. the best paper at NIPS2016, [13]), but has to our knowledge not been done with goal babbling before.

III. Finite Element Goal Babbling

Finite Element Goal Babbling (FEGB) studies in this work, as in the goal babbling framework of [2], the relation between joint angles $q \in \mathcal{Q} \subset \mathbb{R}^m$ and end-effector poses $x \in X \subset \mathbb{R}^n$ of a planar arm. The effector pose is in this case uniquely defined by a forward kinetic function $f(q) = x$, where the aim of traditional goal babbling is in general to find one inverse model $g(x) = q$ such that $f(g(x)) = x$. This is however only useful if the arm is actually able to reach the posture $q = g(x)$, and in many domains (for example when there are obstacles in the task space) this ability depends strongly on the initial state (x_0, q_0) of the arm. To efficiently reach targets x in more complex environments, it is necessary to plan trajectories.

FEGB does this by segmenting the task space into a disjoint set of smaller regions $\{X^{(i)}\}_{i=1}^N$, where each such region is treated as a discrete state $s_i = X^{(i)}$ in a Markov Decision Process (MDP). By learning the probability $P(s, s')$ that the arm can move from a region s to a region s' while using $g(x)$, it is then in turn possible for a higher level *Planner* to plan a sequence of regions to pass through in order to reach any other region with high confidence. Each such transition $s \to s'$ is then physically performed at a lower level by the inverse model (much like an *option* [14]), by choosing a uniformly distributed point x' within the targeted region s' and transform it into a goal posture $q' = g(x')$. Figure 3 gives an overview of the process.

A. Constraints

The abstraction of regions into states of an MDP is useful but puts some constraints on the inverse model, which can be summarized into problems of *consistency*, *division* and *posture search*.

1) The Consistency problem: To be able to plan in the MDP it is to begin with important that there are no hidden variables, see figure 2: Imagine that the inverse model would suggest postures where the arm reaches into s_1 from the right side of the obstacle in some parts of $X^{(1)}$, and from the left side in other parts. It would then be impossible to plan effectively as the planner would need to know if the agent was in s_1 by reaching from the right or left side, in order to know the probability to transition to neighboring states. A solution to this problem is to demand that the inverse model has to be *consistent* within each region, meaning that any two postures q_1, q_2, s. t. $g(q_1), g(q_2) \in X^{(s)}$ can be reached in a direct movement and without leaving s. Going back to figure 2, that means that the inverse model would need to decide on one way to reach region s_1, either only from the left side, or only from the right side. FEGB accomplish this consistency by letting every region s have its independent inverse model $g_s(x)$ based on the observations (x, q) that have previously been made in the state, and that are consistent with previously accepted observations in the state. First observations are always considered consistent. See section III-B for details on how this consistency is determined.

2) The Division problem: A consequence of requiring consistency in each region is that it has to be possible for the inverse model to be consistent within that region. In general this is true if the region is not too big and without obstacles in it. Obstacles thus has to be placed on the edges between different regions. In this work this division is handcrafted.

3) The Posture Search problem: As the dimensionality of the motor space is so much higher than the dimensionality of the task space there is in general an infinite set of possible postures to each region. For efficient control it is however important to notice that not all possible postures within a region are equally capable of moving into neighboring ones. The goal is not only to capture the probabilities of moving between states in the MDP, but also to find the inverse models that maximize these probabilities. To do this it is necessary to allow the inverse models to evolve over time, towards more "useful" postures. Since the inverse models are built on previous observations in the region this posture exploration can be held back by old observations that are increasingly less representative of the current inverse model. For this reason it can be efficient to weigh samples after how recent they are. This can be done in many ways, but in this work this is achieved with a *memory* heuristic. This works so that every new observation is given a value 1 in *memory space*. Every time the observation is then unsed for making an inverse estimate this value is decayed by a small amount

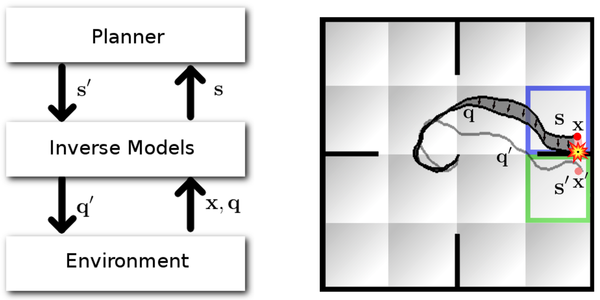

Fig. 3. *Left:* Overview of the proposed system. The inverse models receive from the environment the robot's end-effector position x in task space and its posture q in motor space. The position x is then discretized into the state s depending on the region of x, which is communicated to the planner. The planner then picks a new goal state s' according to equation 8, which is then sent to the inverse models which then picks a random position x' within the region s'. This position x' is then transformed into a goal posture q' using the inverse model of region s'. The arm then tries to do a direct motion into the posture q'. *Right:* An example where the agent starts in state s and attempts a transition to state s'. To get to state s' it attempts posture q', but when trying to get to q' it hits the wall and ends up in task-motor state (x, q), still in state s.

(~ 0.001). This allows the inverse models to identify and give priority to observations that are more recently made, and if the observation was useful, it would have resulted in new observations with similar postures. Observations that do not result in good inverse estimates on the other hand, will fade into oblivion and give room for more useful postures.

Finally are there some cases where an inverse model $g_s(x)$ is found, but where it for some reason becomes unusable. This is rare, but can happen if neighboring regions' inverse models make it impossible to reach any posture $g_s(x)$. Such situations are here dealt with by forgetting that inverse model altogether to allow a more compatible inverse model to be found.

B. Inverse Models

Each inverse model $g_s(x)$ is here based on observations (x_i, q_i) that have previously been made in the region of s. These observations are collected into a dataset $\mathcal{D}_s = \{x_i, q_i\}_{i=1}^{d_s}$, where new observations are only added if they are considered consistent with previously added observations so that the whole dataset is consistent.

1) Generating inverse estimates: Consistency implies that any interpolation $\hat{q}$ of two motor states q_1, q_2 for which $f(q_1), f(q_2) \in s$ is also in s. This allows inverse estimates to be done using for example Linear Regression (LR) [15]. A linear mapping might however not be a good fit at the larger scales of a region, and for that reason are inverse estimates created from subsets $\hat{\mathcal{D}}_{s,x}$ rather than the whole set $\mathcal{D}_s$. The smaller datasets $\hat{\mathcal{D}}_{s,x}$ are formed out of the 100 samples of $\mathcal{D}_s$ that are closest to a target x in combined task- and memory-space, where memory space is seen as just another dimension of task space and where the "memory position" of the target x is set to 1. If the full dataset has less than 100 samples, then $\hat{\mathcal{D}}_{s,x} = \mathcal{D}_s$, unless $\mathcal{D}_s$ is empty. In that case is an extrapolation made from the agents current state s_0 by using $\hat{\mathcal{D}}_{s,x} = \hat{\mathcal{D}}_{s_0,x}$.

This dataset is guaranteed to not be empty, as the arms current state (x_0, q_0) would then be an observation in s_0. Using $\hat{\mathcal{D}}_{s,x}$ it is now possible to estimate a goal posture

$$\tilde{g}_{s,x} = \text{LR}(\hat{\mathcal{D}}_{s,x}), \quad (1)$$

where samples are weighted by their distance to x.

There are now two situations. If the objective is to exploit the inverse model to reach a specific goal, then

$$g^{(s)}(x) = \tilde{g}_{s,x} \quad (2)$$

as it would maximize the chance of reaching x. If the objective on the other hand is to train the inverse model, then some Gaussian noise will be added to the estimate to increase the exploration of the motor space. This gives an estimate

$$g_s(x) = \tilde{g}_{s,x} + rh(P(s_0, s), d_s) \quad (3)$$

where $r \sim U(0,1)$ is a uniformly random scalar, that allows samples to sometimes be close to the best estimate and sometimes more exploratory, and $h(P(s_0, s), d_s)$ is a Gaussian noise that is proportional to the approximated probability $P(s_0, s)$ to reach the state, and inversely proportional to the number of consistent observations d_s made in that state. This allows uncertain transitions to explore the posture space to a greater extent, but also allows the inverse models to converge over time. To add this kind of noise works since only noise that leads to new observations that are consistent with previous data will be added to the datasets, which creates a "survival bias" towards postures that are useful.

2) Determining consistency: In theory it would be necessary to move between all previously seen postures of a dataset to determine if a new observation is consistent with (i. e. inside the convex hull of) these observations. Training data is in this work observations made along the trajectories in task- and motor-space that the agent creates every time it reaches for a goal posture, and these trajectories each consists of 100 data points, unless the movement was interrupted. It would be unfeasible to compare each of these data points to all previous data points in their respective regions. For this paper some assumptions are therefore made to determine consistency more easily. Theoretically there are some situations where these assumptions would accept non-consistent samples, but in practice it seemed to work well enough. The assumptions are:

1) A data point found in a new state is always consistent.
2) It is sufficient that a sample is consistent with one other sample in a dataset, in order to be considered consistent.
3) A posture $q \in g(X^{(s)})$ is consistent with $\mathcal{D}_s$, as it was created as a linear combination of the postures already in that set.

The implication of these rules is (aside from always accepting the first sample in a state) that all directly succeeding samples of a trajectory within a state can be accepted if at least one of the samples is consistent with the dataset of the state. This follows by the fact that it was clearly possible to interpolate between them and the consistent data point, as the whole trajectory is an interpolation between two postures. Practically this applies every time an agent tries to reach a state s and also ends up there in one uninterrupted trajectory, since the last posture of that trajectory would be the goal posture which was generated using the dataset of that state, according to assumption 3).

C. The Planner

The planner has two main purposes. To create a model of the MDP, and to use this model to reach specific goal states. This translates into approximating $P(s, s')$, which is the probability that the agent can move successfully from s to s' if asked to (*explore*), and to plan sequences of states to move through in order to reach a goal state s^* (*exploitation*).

1) Special properties: The MDP has one important hidden variable to keep track of, which is if the agent is in a consistent posture or not. Going back to figure 2, $P(s_1, s_2)$ is here representing the probability that the agent can move from s_1 to s_2 if asked to, *if its initial posture is consistent with the inverse model of* s_1. Assume for example that the arm would end up in s_1 by mistake from the right side of the obstacle, while $\mathcal{D}_{s_1}$ only include left side solutions. The arm would in this case be in an inconsistent configuration, and all previous experiences of transitions from s_1 would be unreliable. This work only considers the arm to be in consistent postures when it ended up in a state that it also tried to reach (see section III-B), which in turn explains why $P(s, s')$ is used and not the more commonly used $P_a(s, s')$, where the probability of ending up in s' when issuing a command a is considered. If the agent would reach for a state in FEGB and not end up there, it is per definition in an inconsistent motor configuration, meaning that any probability estimate starting from that state can not be relied upon. Whenever the agent gets into a non-consistent configuration its main goal is to get back into a consistent one. In general it will then act as it usually would, with a notable exception for the case if it fails to leave its initial region. In that case will it chose the the current state as its next target state. This leads the agent to chose another point x within that state (uniformly) which in turn allows the agent to reposition itself within the region. This is useful since probabilities to leave a state is based on the agent having a random position within the region, and by repositioning itself it makes sure that the probability to leave the state is not biased towards specific positions of that region.

2) Exploration: When estimating the probability of a successful transition: $P(s, s')$, it is important to remember that this probability is based on inverse models that are themselves improving over time and that the rate of this improvement is influenced by how many times the transition has been attempted. This means that if the Planner is selecting a goal state s' in order to estimate $P(s, s')$, it also selects the inverse model $g_{s'}(x)$ to be trained. Because of this the planner has two main challenges when building a model of the MDP:

1) To estimate current transitional probabilities.
2) To decide what inverse models to train.

To this end an intrinsic motivation heuristic, similar to [3], [16] and [17], is introduced. A reward $R(s, s')$ (which is given every time the the agent moves successfully from s to s') is initially set to R_0, and then updated according to

$$R(s, s') \leftarrow \beta R(s, s') \tag{4}$$

every time the transition succeeds. $\beta \in [0, 1]$ is as decay-factor of the reward. To account for the fact that transitional probabilities change over time $P(s, s')$ is estimated using an exponential moving average,

$$P(s, s') \leftarrow \alpha P(s, s') + (1 - \alpha) I(s, s') \tag{5}$$

where $I(s, s')$ equals 1 if the transition is successful and 0 otherwise, and $\alpha \in [0, 1]$ is the decay-factor of previous estimates. Initially all probabilities are optimistically set to $P(s, s') = 1$.

Using $P(s, s')$ and $R(s, s')$ it is possible to compute an estimated return $Q(s, s')$ for *attempting* a transition $s \rightarrow s'$,

$$Q(s, s') = P(s, s')[R(s, s') + \gamma V(s')] \tag{6}$$

where $\gamma \in [0, 1]$ is the future reward discount and

$$V(s) = \max_{s'}\{Q(s, s')\} \tag{7}$$

Where V and Q are estimated using value iteration [15]. Given Q the planner will then choose goal states greedily, so that:

$$s' = \operatorname*{argmax}_{s''}\{Q(s, s'')\} \tag{8}$$

where s'' is limited in the Moore neighborhood of s (including s).

The overall effect of equation (4) and (5) can be seen when considering their effect on the estimated return (6). A transition that is too easy will stop being interesting since $R \rightarrow 0$ (unless the transition opens up for delayed future reward), and a transition that is too hard will stop being interesting since $P \rightarrow 0$. This makes the agent focus on the transitions in-between, where it has some probability of success but where only a few successes have previously been seen. These are also transitions where the inverse models have good opportunities for improvement since transitions are clearly possible, but not yet performed in a reliable way.

The estimated reward of one attempted transition $s \rightarrow s'$ is $P(s, s')R(s, s')$, and this reward will always decrease if the transition fails, as P decreases while R stays fixed. If the transition is a success on the other hand, then P will increase and R will decrease. In particular if $P(s, s') = P$ and $R(s, s') = R$ before a success, and R', P' after, then

$$R' = \beta R \tag{9}$$

and

$$P' = \alpha P + (1 - \alpha) \tag{10}$$

Combining (9) and (10) it is possible to show that the estimated one step reward $P(s, s')R(s, s')$ stays fixed for an initial probability $P = P^*$ where

$$P^* = \frac{(1 - \alpha)\beta}{1 - \alpha\beta} \tag{11}$$

If $P < P^*$ before the transition, $P(s, s')R(s, s')$ will increase in case of success. This creates a mechanism that fundamentally controls the balance between the two objectives of the planner when building a model. A low P^* results in the planner pushing the agent to attempt more unlikely transitions, since all probabilities $P > P^*$ will decrease in estimated return after a success, while a high P^* encourages the agent to explore all transitions more evenly, creating a better model, but without pushing the inverse models as hard to improve.

For this work $P^* = 0.3$ and $\alpha = 0.8$, giving $\beta \approx 0.68$. The future reward discount was set to $\gamma = 0.95$.

3) Exploitation: Once a model is built it can be used to plan trajectories to any state/region s of the task space. Assume that a goal position $x^* \in s^*$ is given (e.g. by an external user). It is then up to the planner to find a sequence of transitions that is both short and likely to get the agent to s^*. This is done by redefining the rewards $R(s, s')$ so that

$$R(s, s') = \begin{cases} 1 \quad , \quad s' = s^* \\ 0 \quad , \quad otherwise \end{cases} \tag{12}$$

This reward inserted in equation (6) of Q will lead the planner to consistently choose the goal states s' with the highest probability of reaching s^* while penalizing longer paths due to future reward discount γ.

IV. Experiments

A. Setup

The environment illustrated in Figure 1 is used to evaluate the performance of FEGB. This setup includes an arm of length 1 and width 0.01 units, in an enclosed square room with sides of 1 unit and with walls of length 0.2 reaching into the room. The base of the arm is fixed to the middle of the room and has uniformly spaced joints where the number of joints depends on its degrees of freedom. Every joint can be given any angle $[-\pi, \pi]$, and the arm will move towards any given motor-goal in 100 steps in a linear fashion in motor space. This movement will be stopped at any point if the subsequent position would break the physical laws of the environment, which consists of:

- The arm cannot pass trough walls.
- The arm cannot pass through itself.
- The arm can not move infinitely fast. This is in practice accomplished by restricting the speed of any joint to be lower than some threshold, which limits the magnitude of the arm's movement at each iteration.

This setup provides a complex problem where a high dimensional motor space makes most traditional methods unfeasible for rapid online learning, while environmental restrictions break vital assumptions for using goal babbling.

Two measures are used in evaluating the method:

- Inverse model error
- Reaching error

The inverse model error ϵ_{inv} measures the distance

$$\epsilon_{inv} = ||x^* - f(g(x^*))||_2 \tag{13}$$

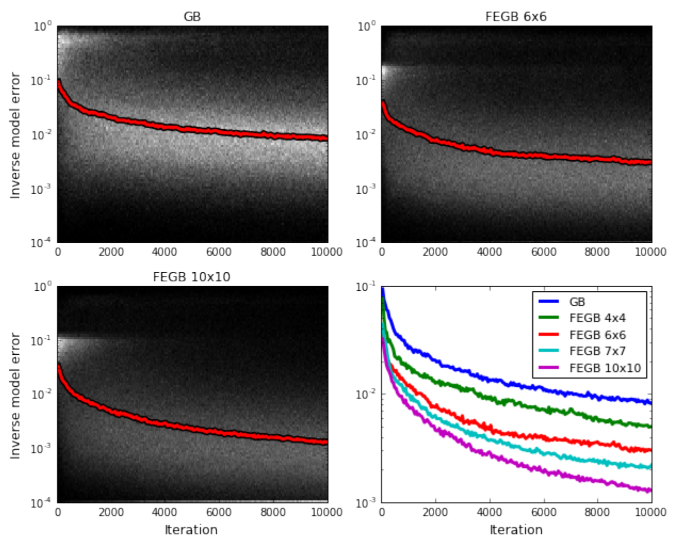

Fig. 4. The inverse error over time for different resolutions of subtask-spaces and 100 degrees of freedom. The background shows the density of samples, where lighter colors signify higher concentration. Each column of pixels represents the density of 20×200 evaluations (20 independent runs, each evaluated 200 times). The lines show the geometrical mean of the distributions to highlight the average dimension of the error.

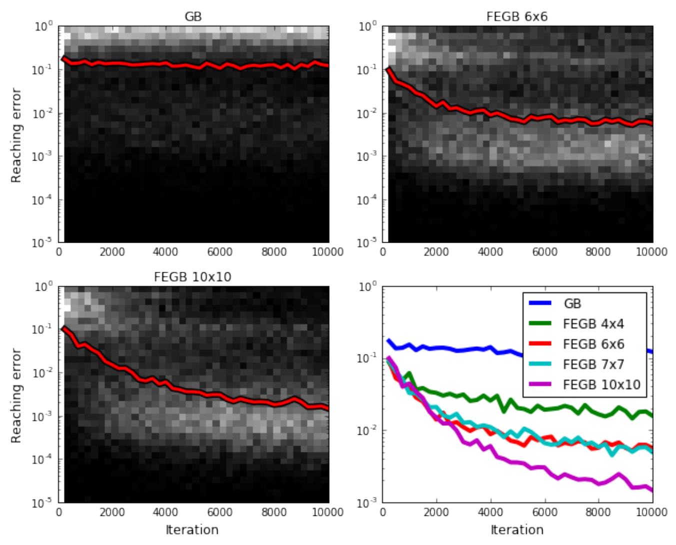

Fig. 5. The reaching error over time, for different resolutions of subtask-spaces and 100 degrees of freedom. Each column of pixels represents the density of 20×100 evaluations (20 independent runs, each evaluated 100 times). The lines show the geometrical mean of the distributions.

between a randomly sampled task goal x^* and the end effector position of the agent given its inverse estimate $q^* = g(x^*)$, ignoring whether q^* can be physically reached or not. This is computed off-line by simply applying the forward kinematics $f(q^*)$. If x^* is in a state for which there is no inverse model, it will be re-sampled.

The Reaching error ϵ_{reach} measures how close the agent can get to a random goal x^* when starting in a motor state $q_0 = g(x_0)$, where x_0 is another random point in the task space. The agent is then allowed multiple steps in the MDP for reaching s^* for which $x^* \in s^*$, leading to a trajectory $\{x_t\}_t^T$ in the task space. Using this trajectory the reaching error is defined as:

$$\epsilon_{reach} = \min_t ||x_g - x_t||_2 \tag{14}$$

This measure is used so that a measurement will be given even when the arm fails to reach the goal state and therefore never stops moving.

FEGB is evaluated for:

- Different resolutions when partitioning X, where a resolution of 1×1 is equivalent to goal babbling, as it does not allow for planning.
- Different numbers of degrees of freedom of the arm.
- Effect of extended training-time.

The method is here only compared to one version of goal babbling. The important thing to remember is that FEGB is different from goal babbling in that it adds a planing layer. The emphasis is therefore not on how fast and well an inverse model can be learned by a goal babbling architecture, but on showing the limitation of only building an inverse model without allowing for planning.

Every setting is evaluated for 20 independent runs and for 10000 iterations each (except in the extended training-time case). The inverse error is computed every 50th iteration by collecting 200 samples, while the reaching error is computed every 250th iteration by collecting 100 independent samples. In the extended training-time test only one run is made. It runs for 50000 iterations and 200 inverse error values are collected every 250th iteration and 100 reaching errors are collected every 1000th iteration.

B. *Results*

Figures 4 and 5 show the effect of different resolutions when partitioning the task-space. Figure 6 shows the effect of different degrees of freedom, Figure 7 the effect of extended training-time and Figure 1 compares some trajectories developed in the different settings. Most significant in Figure 1 is that the arm is unable to reach a goal on the other side of a wall without the ability to plan, as it tries to go straight for the goal posture even though the wall stops it from actually reaching it.

1) Resolution - GB vs FEGB: It is clear that the added planning capability of FEGB substantially increases the performance in terms of both the errors, considering Figures 4 and 5. Especially in terms of the reaching error does the advantages of FEGB over GB become clear, which is expected since GB is not designed for scenarios where planning is necessary. This can be seen in particular by noting that the inverse errors and the reaching error are roughly the same for FEGB, while GB has a better inverse error than reaching error, meaning that it is unable to reach its goal postures. Generally it seems like the higher resolution the better but this trend must eventually break since a higher resolution means more inverse models to train and a more complex MDP model to learn. These factors

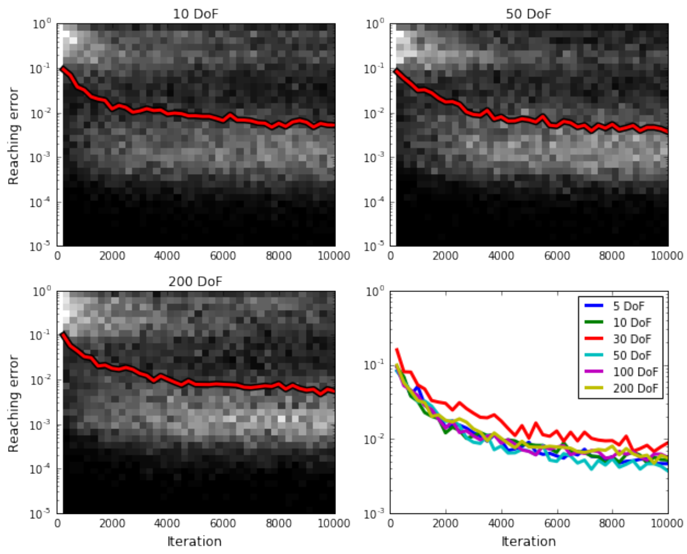

Fig. 6. The reaching error over time, for different degrees of freedoms of the arm and resolution 6×6. Each column of pixels represents the density of 20×100 evaluations (20 independent runs, each evaluated 100 times). The lines show the geometrical mean of the distributions.

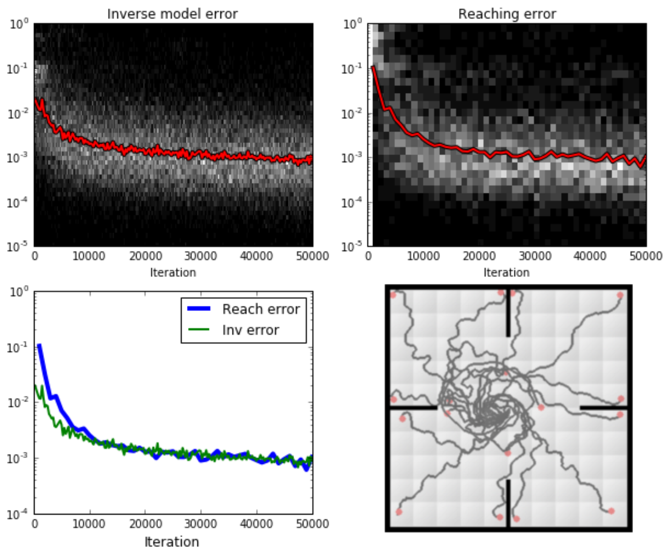

Fig. 7. The result of a single longer run with resolution 10×10 and 100 degrees of freedom. Each column of pixels represents the density of 200 samples for the inverse model error, and 100 samples for the reaching error. The lines show the geometrical mean of the distributions.

should make the training time longer but it seems like 10×10 is not big enough for this effect to become significant. What is interesting is that 7×7 is managing pretty well considering that the odd number will place some walls straight through some of the states, making it impossible to find consistent datasets for the whole region. By manually examining the trajectories made in this case we hypothesize that the planner learns to avoid these states by primarily planing its trajectories through other more reliable, states.

2) Degrees of freedoms: From the results of Figure 6 it appears that the degrees of freedom has little or no effect on the effectiveness of the method. This is in line with previous results from goal babbling [2] and shows that this property transfers to FEGB.

3) Extended training-time: Figure 7 shows that the method is stable and that the inverse error and the reaching error are virtually identical after approximately 10000 iterations, meaning that the agent can basically move between any two positions in the task space at that point.

V. Discussion

Finite Element Goal Babbling provides a framework for rapid online learning of high dimensional systems in lower dimensional task-space where the reachability of a position in task- or motor-space is dependent on the initial condition of the system. It is thus extending goal babbling to a wider set of problems while clearly keeping its indifference to the motor-space dimensionality and also preserves a rapid learning rate. These properties make it an interesting candidate for on-line learning in physical robots, as it is applicable to high degrees of freedoms while avoiding limitations and problems of other state-of-the-art methods, such as the long training time and large databases needed for most Deep Reinforcement Learning (DRL) solutions [18], [19], [20]. Additionally it also provides a user control over the trained robot since goal states can be given from outside. It however lacks the ability to process large sensor spaces like in DRL, and it could be very interesting to see if there are possible fusions of the two approaches. One promising approach within the field of goal babbling of how to work with larger task spaces can be found in [6], where a goal babbling architecture is used to solve multiple task-spaces in parallel.

Another important topic to study is methods for the agent to partition the task space autonomously (as is done in [10] e.g.), as the structure of the environment is often not known it advance. A more exhaustive investigation into the intrinsic motivation of the agent could also be beneficial since there exists many different promising implementations, even though such a comparison was outside the scope of this work.

Finally it would be interesting to study FEGB in environments that shape the motions of the agent to a wider extent. The current environment is very unforgiving in the sense that a motion is either entirely allowed, or entirely refused. A more “shaping” environment could maybe help bootstrapping the search for efficient motions and postures of the agent, an effect shown in [21] among other works.

Acknowledgments

This project has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 674868 (APRIL).

References

[1] A. Cangelosi and M. Schlesinger, *Developmental robotics: From babies to robots*. MIT Press, 2015.

[2] M. Rolf, J. J. Steil, and M. Gienger, "Goal babbling permits direct learning of inverse kinematics," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 3, pp. 216–229, 2010.

[3] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, Apr. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4141061

[4] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (19902010)," *Autonomous Mental Development, IEEE Transactions on*, vol. 2, no. 3, pp. 230–247, 2010.

[5] K. Narioka, R. F. Reinhart, and J. J. Steil, "Effect of exploratory perturbation on the formation of kinematic synergies in Goal Babbling," in *2015 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, Aug. 2015, pp. 86–91.

[6] S. Forestier and P.-Y. Oudeyer, "Modular active curiosity-driven discovery of tool use," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 3965–3972.

[7] C. Moulin-Frier, S. M. Nguyen, and P.-Y. Oudeyer, "Self-organization of early vocal development in infants and machines: the role of intrinsic motivation," *Frontiers in Psychology*, vol. 4, 2014. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fpsyg.2013.01006/abstract

[8] A. Philippsen, F. Reinhart, and B. Wrede, "Goal babbling of acoustic-articulatory models with adaptive exploration noise," 2016.

[9] F. Benureau, "Self-Exploration of Sensorimotor Spaces in Robots," phdthesis, Universit de Bordeaux, May 2015. [Online]. Available: https://hal.archives-ouvertes.fr/tel-01251324/document

[10] A. Baranes and P.-Y. Oudeyer, "Active learning of inverse models with intrinsically motivated goal exploration in robots," *Robotics and Autonomous Systems*, vol. 61, no. 1, pp. 49–73, Jan. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0921889012000644

[11] M. Rolf and J. J. Steil, "Efficient Exploratory Learning of Inverse Kinematics on a Bionic Elephant Trunk," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 6, pp. 1147–1160, Jun. 2014.

[12] C. Moulin-Frier, P. Rouanet, and P.-Y. Oudeyer, "Explauto: an open-source python library to study autonomous exploration in developmental robotics." IEEE, pp. 171–172. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6982976

[13] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, pp. 2154–2162. [Online]. Available: http://papers.nips.cc/paper/6046-value-iteration-networks

[14] B. Hengst, "Hierarchical Approaches," in *Reinforcement Learning*, M. Wiering and M. van Otterlo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 12, pp. 293–323. [Online]. Available: http://link.springer.com/10.1007/978-3-642-27645-3_9

[15] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[16] M. Frank, J. Leitner, M. Stollenga, A. Frster, and J. Schmidhuber, "Curiosity driven reinforcement learning for motion planning on humanoids," *Frontiers in Neurorobotics*, vol. 7, 2014. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnbot.2013.00025/abstract

[17] C. Salge, C. Glackin, and D. Polani, "Empowerment – an Introduction," *arXiv:1310.1863 [nlin]*, Oct. 2013, arXiv: 1310.1863. [Online]. Available: http://arxiv.org/abs/1310.1863

[18] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv:1506.02438 [cs]*, Jun. 2015, arXiv: 1506.02438. [Online]. Available: http://arxiv.org/abs/1506.02438

[19] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *arXiv:1504.00702 [cs]*, Apr. 2015, arXiv: 1504.00702. [Online]. Available: http://arxiv.org/abs/1504.00702

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[21] R. Der and G. Martius, "Novel plasticity rule can explain the development of sensorimotor intelligence," *Proceedings of the National Academy of Sciences*, vol. 112, no. 45, pp. E6224–E6232, Nov. 2015. [Online]. Available: http://www.pnas.org/lookup/doi/10.1073/pnas.1508400112