Combining Gaussian Processes and Conventional Path Planning in a Learning from Demonstration Framework

Markus Schneider, Richard Cubek, Tobias Fromm, and Wolfgang Ertel

Autonomous Mobile Service Robots Laboratory
Ravensburg-Weingarten University of Applied Sciences
Doggenriedstrasse, 88250 Weingarten, Germany
{firstname.lastname}@hs-weingarten.de
http://www.zafh-servicerobotik.de/en

Abstract. Today, robots are already able to solve specific tasks in laboratory environments. Since everyday environments are more complex, the robot skills required to solve everyday tasks cannot be known in advance and thus not be programmed beforehand. Rather, the robot must be able to learn those tasks being instructed by users without any technical background. Hence, Learning from Demonstration (LfD) is one of the essential topics to bring robots out of the lab moving towards everyday robustness. The key property of an agent regarding a demonstration learned skill is its ability of generalization, that is, applying a learned skill to situations that differ from those during demonstration. In this paper, we present a method to enhance the generalization capabilities of an advanced new LfD framework by combining it with conventional path planning.

Key words: Learning from Demonstration, Programming by Demonstration, Gaussian Processes, Path Planning

1 Introduction

Preparing robots to face everyday environments requires them to have a set of predefined abstract skills, where abstraction is needed to generalize over similar tasks. But is it possible to predefine all skills a robot needs during lifetime? For example, setting the table consists of several simple *pick and place* tasks, which the robot should be able to handle natively. But additionally, there are underlying constraints such as the necessity of placing saucers on the table before placing the cups, to mention only one.

Taking a closer look at the problem, the conclusion is that it is impossible to predefine all needed skills for complex everyday environments. Robots will have to acquire them during lifetime fulfilling the requirement of reproduction in situations that differ from those during demonstration. Such a skill aquisition could be realized in different ways. Programming or describing by any kind of programming or description language is not suitable for daily use, especially

not for users without profound computer knowledge. Verbal robot instruction is an interesting, but comparatively small research area [Bug03] and should be treated as an extension to other techniques. Since even humans tend to learn by following examples rather than verbal instructions, many research groups follow a Learning from Demonstration (LfD) approach. This important field is also addressed by on of the the robocup@home tasks, the *CopyCat* game, where the robot has to reproduce a human demonstrated block movement in a game-like setting.

1.1 Learning from Demonstration: Different Approaches

In general, the representation of a skill can take place on two abstraction layers: a low level representation *(trajectory encoding)* for generic motions and a high level representation *(symbolic encoding)* for sequences of predefined actions [BCDS08]. Trajectory encoding allows us to describe arbitrary motor primitives and during training very often requires direct motion of the robot's actuator by a human trainer. High-level learning requires a predefined set of low-level skills, but allows the description of a more abstract action sequence for more complex, goal oriented tasks. LfD can address different abstraction layers. For instance, a high-level learning approach can be based on the detection of spatio-temporal task constraints, as presented in [EK08].

Since motor primitives have to be mastered before high-level skills can be applied, most works are located in the area of low-level skills. A popular approach is a demonstration of a motion by a human, followed by an optimization via Reinforcement Learning. This area of research is mainly focused on advanced robot actuators of high mechanical flexibility for handling low-level tasks which are very difficult to control even for humans. Interesting examples are the *ball in cup* [KMP08] and *t-ball batting* [PS07] problems.

There are also approaches to develop an integrated overall solution as presented in [EZRD02]. A demonstration is observed by a set of distributed sensors, recording motions (including graspings) as well as object positions to allow a detailed analysis of the human behavior. Recognizing elementary actions, the demonstration is divided into semantically related segments and mapped onto a sequence of predefined symbols. After further processing, this abstract representation can be mapped onto low-level skills of suitable robotic systems realizing both, reproduction of complex skills on changing environments and changing robotic embodiments.

Another powerful technique is the low-level time-depending observation of the actuator's end effector relative to each object in the task [Cal07,CB08]. After a couple of demonstrations, the time-depending variations between demonstrations are modelled by Gaussian Mixture Regression (GMR). In the reproduction phase, the generated trajectories relative to each object are weighted anti proportional to their variance. The reproduction can now be generated in compliance to the variations, successfully handling new situations where the objects can be placed at arbitrary positions. The first part of our work is based on this approach with the difference that we use Gaussian Processes (GP) instead of GMR.

1.2 Generalization Issue

Beside robustness, a demonstration learned behavior is evaluated by its applicability to situations that differ from those during demonstration. The abstraction capability of a GP based LfD framework is strong. After three or four demonstrations, the robot will solve the demonstrated task successfully in a scene with objects being repositioned arbitrarily. As a weakness of this method, the generalization capability ends at the point where objects or obstacles occur which have not been present or relevant during demonstration. This kind of objects is ignored by the motion plan what may cause collisions. We address this problem with our extended and combined method. It enhances the learned skill by expanding the class of situations the skill can be applied to.

2 Methods

2.1 Gaussian Process Models

In recent years, Gaussian Processes became very popular in the context of machine learning for regression and classification problems [RW06]. They are powerful tools to solve non-linear problems while requiring relatively simple linear algebra only. The Gaussian Process theory provides a very natural way to define a prior distribution over (regression) functions. In the standard non-linear regression problem, we try to estimate a latent function $f(\boldsymbol{x})$ given input variables $\boldsymbol{x} \in \mathbb{R}^D$ and noisy observed target values $y \in \mathbb{R}$ modeled as $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \in \mathbb{R}$ is a random noise variable that is independent, identically distributed (i.i.d.) for each observation. The training data set comprising n input points together with the corresponding observations is denoted by $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)|_{i=1}^n\}$. The Gaussian Process regression model tries to learn the predictive distribution $p(f^*|\boldsymbol{x}^*, \mathcal{D})$ of a new test output f^* given a test input $\boldsymbol{x}^*$. To simplify notation, all training inputs $\{\boldsymbol{x}_i\}_{i=1}^n$ are collected in a so called *design matrix* $\boldsymbol{X}$ of size $D \times n$ and we define the matrix $\boldsymbol{X}^*$ and the vectors $\boldsymbol{f}^*, \boldsymbol{y}$ in the same way. The key in Gaussian Processes is to consider the training outputs $\boldsymbol{y}$ and the new testing points (prediction values) $\boldsymbol{f}^*$ as a sample from the same (zero mean) multivariate Gaussian distribution. The predictive distribution is then again a multivariate Gaussian distribution $\mathcal{N}(\bar{\boldsymbol{f}}^*, \mathrm{cov}[\boldsymbol{f}^*])$ conditioned on the training data with mean

$$\bar{\boldsymbol{f}}^* = \boldsymbol{K}^*(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{y} \tag{1}$$

and covariance matrix $\mathrm{cov}[\boldsymbol{f}^*]$

$$\mathrm{cov}[\boldsymbol{f}^*] = \boldsymbol{K}^{**} - \boldsymbol{K}^*(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1}\boldsymbol{K}^{*T}, \tag{2}$$

where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{K} \in \mathbb{R}^{n \times n}$, $\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{K}^* \in \mathbb{R}^{n^* \times n}$, $\boldsymbol{K}^*_{i,j} = k(\boldsymbol{x}^*_i, \boldsymbol{x}_j)$ and $\boldsymbol{K}^{**} \in \mathbb{R}^{n^* \times n^*}$, $\boldsymbol{K}^{**}_{i,j} = k(\boldsymbol{x}^*_i, \boldsymbol{x}^*_j)$. The function $k(\cdot, \cdot)$ is called covariance function, or kernel, and is used to construct the covariance matrices $\boldsymbol{K}, \boldsymbol{K}^*, \boldsymbol{K}^{**}$. We will write the Gaussian Process as $\mathcal{GP}(\boldsymbol{0}, k(\cdot, \cdot))$ or simply $\mathcal{GP}$.

Typically, the covariance function depends on parameters $\boldsymbol{\omega}$ which are called *hyper parameter* because they are determined in advance and not used to absorb the training data information. A widely used covariance function is given by the exponential of a quadratic form, namely the *Squared Exponential* (SE) to give

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2}{2l^2}\right), \tag{3}$$

with $\boldsymbol{\omega} = (\sigma_n, \sigma_f, l)^T$ defining the *noise level*, *signal variance* and *characteristic length scale* respectively. We use $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\omega})$ to obtain the *log marginal likelihood* given by

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\omega}) = -\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}_y^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}_y| - \frac{n}{2}\log 2\pi, \tag{4}$$

where $\boldsymbol{K}_y$ substitutes $\boldsymbol{K}+\sigma_n^2\boldsymbol{I}$ and $|\cdot|$ is the determinant. We then use a *conjugate gradient* algorithm [NW99] to (locally) maximize the log marginal likelihood function with respect to the hyper parameters.

The Gaussian Process Model described so far assumes a constant noise level. Given a data set that requires a variable noise model would be able to correctly estimate the mean value, but it would fail to correctly estimate the variance. We therefore have to introduce a more flexible noise model called *heteroscedasticity*. Replacing the constant noise rate σ_n by an input dependent noise function $r(\boldsymbol{x})$, the mean and covariance function of the predictive distribution changes to

$$\bar{\boldsymbol{f}}^* = \boldsymbol{K}^*(\boldsymbol{K} + \boldsymbol{R})^{-1}\boldsymbol{y} \quad \text{and} \tag{5}$$

$$\operatorname{cov}[\boldsymbol{f}^*] = \boldsymbol{K}^{**} + \boldsymbol{R}^* - \boldsymbol{K}^*(\boldsymbol{K} + \boldsymbol{R})^{-1}\boldsymbol{K}^{*T} \tag{6}$$

respectively. $\boldsymbol{R}$ is defined as $\operatorname{diag}(\boldsymbol{r})$, with $\boldsymbol{r} = (r(\boldsymbol{x}_1), \ldots, r(\boldsymbol{x}_n))^T$, and $\boldsymbol{R}^* = \operatorname{diag}(\boldsymbol{r}^*)$, with $\boldsymbol{r}^* = (r(\boldsymbol{x}_1^*), \ldots, r(\boldsymbol{x}_{n^*}^*))^T$. We follow [GWB97] and use a second independent Gaussian Process (the z-process denoted by $\mathcal{GP}_z$) to model the log of the noise level giving $z(\boldsymbol{x}) = \log(r(\boldsymbol{x})) = (z_1, \ldots, z_n)^T$. The z-process maintains its own independent covariance function $k_z(\cdot,\cdot)$ and parameters ω_z. With $\boldsymbol{z}^*$ as the $\mathcal{GP}_z$ posterior prediction, we obtain the predictive distribution $p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}) =$

$$\int\int p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}, \boldsymbol{z}, \boldsymbol{z}^*)p(\boldsymbol{z}, \boldsymbol{z}^*|\boldsymbol{X}^*, \mathcal{D})\, d\boldsymbol{z}\, d\boldsymbol{z}^*, \tag{7}$$

where the last term prevents an analytical solution of the integral. A common method is to approximate $p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}) \approx p(\boldsymbol{f}^*|\boldsymbol{X}^*, \mathcal{D}, \tilde{\boldsymbol{z}}, \tilde{\boldsymbol{z}}^*)$, where $\tilde{\boldsymbol{z}}, \tilde{\boldsymbol{z}}_* = \arg\max_{\boldsymbol{z},\boldsymbol{z}^*} p(\boldsymbol{z}, \boldsymbol{z}^*|\boldsymbol{X}^*, \mathcal{D})$. We use the expectation-maximization (EM) algorithm, introduced by [KPPB07], to iteratively estimate $\mathcal{GP}_z$ and then combine it with a noise free $\mathcal{GP}$ to estimate a heteroscedastic Gaussian Process. The whole procedure follows the EM algorithm for the heteroscedastic Gaussian Process Model as described in [KPPB07].

2.2 The GP based Learning from Demonstration Framework

We use the same learning from demonstration framework as described in [CB08] to encode a task and summarize the used methods. During a demonstration, sensor information is collected together with a timestamp and stored as a state/observation $\boldsymbol{\xi} = (\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) \in \mathbb{R}^D$. Here, $\xi_t^{(i)}$ denoting the i-th *temporal value* $\in \mathbb{R}$ and $\boldsymbol{\xi}_s^{(i)} \in \mathbb{R}^{(D-1)}$ is the i-th vector of *spatial values*. A mapping from $\xi_t^{(i)}$ to $\boldsymbol{\xi}_s^{(i)}$ is called a *policy* and denoted with π. In our experiments we use a simple timestamp t as the temporal value and the spatial value is represented through coordinates in *joint space* ($\theta_1, \ldots, \theta_k$ for the k motor encoders of the robot) or *task space* (Cartesian coordinates and orientation of the end effector). Suppose we have n demonstrations and each demonstration is resampled to a fixed size of T, then the data set $\mathcal{D}$ of all observations will be of length $N = nT$, formally $\mathcal{D} = \{(\xi_t^{(i)}, \boldsymbol{\xi}_s^{(i)}) | i = 1, \ldots, N\}$. The policy used here maps from ξ_t to $\boldsymbol{\xi}_s$ using $p(\boldsymbol{\xi}_s | \xi_t, \mathcal{D}) = \mathcal{GP}(\cdot, \cdot) \, \forall \, \xi_t \in \mathbb{R}$ in terms of a Gaussian Process. This approach perfectly fits the requirements of an approximation in the context of robots, namely generating *continuous and smooth paths* and provide a *generalization over multiple demonstrations*. The Gaussian Process covariance function controls the function shape and ensures a smooth path and the fundamental GP algebra calculates the mean over demonstrations. This reduces noise (introduced by sensors and human jitter during the demonstration) and recovers the underlying trajectory.

The key idea is to use the variations and similarities between demonstrations to extract *what* is important to imitate. Assume a robot arm scenario where the goal is to reach a specific end state (e.g. given 3D coordinates) and the start state is chosen randomly. The trajectories will differ significantly at the beginning, whereas the position should be more or less the same at the end. Such a part of a trajectory with low variation is defined as a *constraint* because no discrepancy is desired. The *heteroscedastic* Gaussian Process Model, as discussed in section 2.1, allows us to encode both the trajectory and the variation between the demonstrations.

In the training (encoding) phase, policies are created with respect to the (absolute) joint angle trajectories θ and relative to all $m \in M$ objects $\boldsymbol{o}^{(m)}$ detected in the environment. More precisely, the relative position of the robot's end effector to the initial position of each object (in Cartesian coordinates) is calculated.

In the reproduction phase, a Cartesian trajectory estimation $\hat{\boldsymbol{x}}^{(m)}$ and a related covariance matrix $\hat{\boldsymbol{\Sigma}}^{x(m)}$ for each detected object[1] is calculated by the Gaussian Process policies, giving M constraints. These constraints are used together with the joint angle estimation $\hat{\boldsymbol{\theta}}$ and related covariance matrix $\hat{\boldsymbol{\Sigma}}^{\theta}$ to create a final policy which considers constraints in both, Cartesian space and joint space. The projection from Cartesian space to joint space is done with a pseudoinverse Jacobian algorithm that compensates the missing orien-

[1] It is no problem if some of the objects from the training phase are not detected. Constraints with respect to these objects will be skipped.

tation information of the end effector [Lie77,CB08]. This can be accomplished through an optimization of the Jacobian null space with respect to the demonstrated joint angle trajectories (stay as close as possible to $\hat{\boldsymbol{\theta}}$). The projection is applied to both the trajectory estimations $\hat{\boldsymbol{x}}^{(m)}$ and the covariance matrices $\hat{\boldsymbol{\Sigma}}^{x(m)}$ giving m additional joint space estimates $\hat{\boldsymbol{\theta}}^{(m)}$, $\hat{\boldsymbol{\Sigma}}^{(m)}$. To retrieving a generalized version of the trajectories, two distributions $\mathcal{N}(\hat{\boldsymbol{\theta}}^{(i)}, \hat{\boldsymbol{\Sigma}}^{(i)})$ and $\mathcal{N}(\hat{\boldsymbol{\theta}}^{(j)}, \hat{\boldsymbol{\Sigma}}^{(j)})$ are assumed as independent and the Gaussian product property is used as $\mathcal{N}(\boldsymbol{\theta}, \boldsymbol{\Sigma}) = \mathcal{N}(\hat{\boldsymbol{\theta}}^{(i)}, \hat{\boldsymbol{\Sigma}}^{(i)}) \cdot \mathcal{N}(\hat{\boldsymbol{\theta}}^{(j)}, \hat{\boldsymbol{\Sigma}}^{(j)})$. It is not necessary to perform the Gaussian product at each time step of the trajectory, because the Gaussian Process already delivers the covariances between all data points of the whole trajectory. All steps described here follow the procedure from [CB08,Cal07] with the option to consider constraints in *joint space only*, *task space only* or *joint and task space*.

Figure 1 shows the demonstration of a *grasping and emptying* task. The resulting policies generated by the GP based framework are shown in Fig. 2.

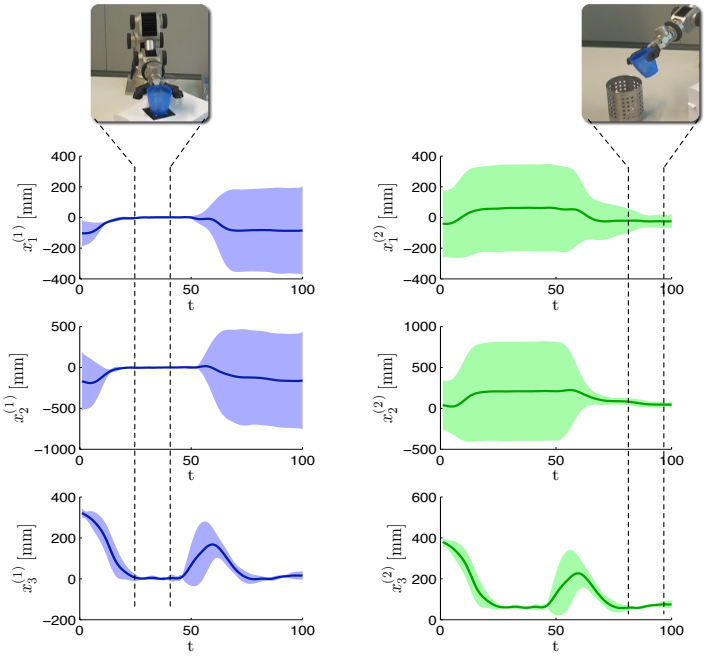
(a) (b) (c)

Fig. 1: Demonstration of the *grasping and emptying* task. The first subgoal is to move the robot arm in proper position in order to grasp the blue cup (a). Subsequently, the arm has to move slightly besides the trash (b) to empty the cup into the trash (c).

2.3 Conventional Path Planning

So far, we considered learning from demonstration. A totally different problem is given in a situation where the robot's actuator starting pose and the desired goal pose are already known in advance. Here, all we have to do is to find a path from the start to the goal while avoiding obstacles. Path planning is the search for a continuous sequence of actuator joint configurations between an initial start configuration and a final goal configuration under the compliance of certain constraints. The term *motion planning* refers to the same problem, but usually describes a path parametrized by time (i.e. a trajectory).

There are several algorithms realizing path or motion planning by heuristic search algorithms. One example are *Rapidly-exploring Random Trees* (*RRT-*



(a) Relative to cup (b) Relative to trash

Fig. 2: The learned policies from a *grasping and emptying* task (similar to Fig. 1) relative to the initial positions of the cup (a) and the trash (b) in Cartesian coordinates. We can see that the movement relative to the cup is highly constrained between time steps 20 and 40. At this time, the end effector reaches the cup and grasps it. In contrast, the movement relative to the trash is constrained at the end of the trajectory (time steps 85 to 100), when the arm is able to unload the cup.

Connect) where the search space is explored from both directions, the start and the goal configuration, moving towards each other using a greedy heuristic [KSL00]. We use this algorithm due to its speed and accuracy. It is available within OpenRAVE, a powerful planning environment for robotics, supporting most known robot arms and also providing visualization, simulation and scripting interfaces [DK08]. Furthermore, OpenRAVE offers the possibility to search for paths not only avoiding obstacles but also considering arbitrary, self-programmed path constraints. For example, we can place a control function in the planner which will be called on every search step. This function will be allowing an explored joint configuration only if the end effector keeps a horizontal position, achieving a trajectory suitable to move a cup filled with a liquid. OpenRAVE also includes inverse kinematic solvers, offering the calculation of possible joint configurations for given end effector poses.

2.4 Combining both methods

An LfD oriented, GP based motion planning approach, as it was introduced in 2.2, and the conventional approach described in 2.3 cannot be compared directly because they address different problems. In the LfD framework, we have to extract constraints from demonstrations. Then, at the starting position when applying a skill, the problem is to generate a trajectory considering these constraints in a new situation. Regarding classical path planning, at the starting position we already know the initial and final joint configurations and the problem is to find a trajectory avoiding obstacles. What the methods have in common is the type of result, they both generate trajectories.

A trajectory from the LfD framework was built considering variances related to objects being present during demonstrations, but ignoring new objects in new situations. On the other hand, a classically generated trajectory is only considering the current state, but with all objects in the scene. It is obvious that the strengths can be combined.

Taking a closer look on the motion plan from the example in Fig. 3, we can see that there are low constrained regions (large variance) at the beginning and between time steps 60 and 80 regarding both objects (cup and trash). They refer to the first motion towards the cup and the motion towards the trash after the cup was grabbed. These are small regions regarding time, but they refer to the largest spatial motions. Hence, they represent the regions of highest collision risk regarding unknown objects. This is not an interpretation of this special example only. It is typical that we have large variances at regions of large spatial motions and small variances for motions of less spatial characteristics. A region of large variance means that it is low constrained and that we could move on many different paths to fulfill the required needs in the concerned segment. This is the most important property exploited in our new approach. Receiving a trajectory from the LfD framework, we cut out the regions of high variance with the intention to plan a new trajectory for these segments. Since we know the joint configurations at both ends of such a segment, we simply use classical path search avoiding all obstacles within the scene. The result is a powerful LfD framework, handling constraints from demonstrations as well as unknown objects in new situations.

At some paths, when low constrained regions are reached, a GP based motion still keeps constraints like holding a cup in a horizontal position due to the fact that the path is averaged over the demonstrations. Cutting the path at such a position and handing over responsibility to OpenRAVE could cause a subsequent plan not taking care of the end effector's angles, especially when obstacles have to be avoided. Therefore, at the end of a highly constrained path, the end effector's pitch and roll angles are read and a path planning constraint function in OpenRAVE is initialized, effecting OpenRAVE to keep up the mentioned angles.

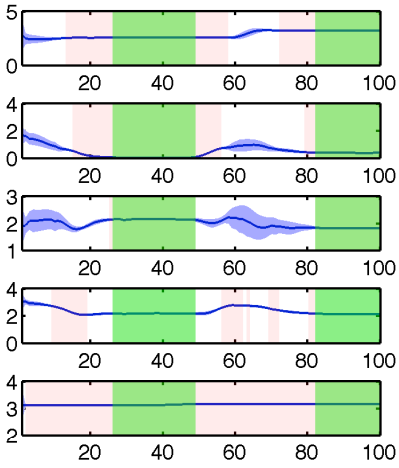
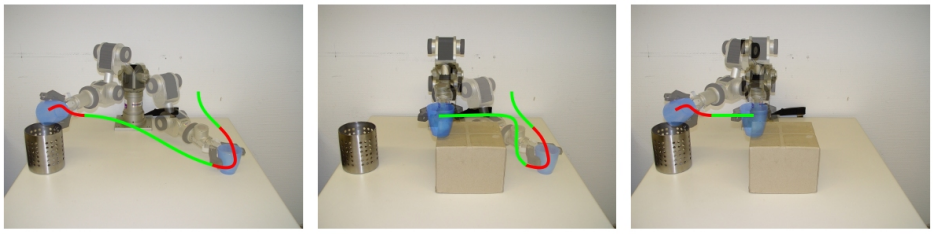


Fig. 3: The overall policy plotted for each DOF against time. This is the result of merging policies related to individual objects (as shown in Fig. 2). In the merging process, the smallest variances have the highest weights. Regions with low variance in each DOF independently are marked red. Green denotes the intersection of low variance regions for all DOFs. We cut out all non-green segments and generate new trajectories there with OpenRAVE. Highly constrained regions (green) remain as they are, since they represent the compliance of constraints learned from demonstration.

3 Results and Discussion

We tested the method learning *grasping and emptying* and a simple *pick and place* task, each demonstrated four times on a *Neuronics Katana* arm. In situations where the learned skill had to be applied to, all known objects have been repositioned and one or two new objects have been added to the scene. The method performed well reproducing demonstrated skills considering new objects successfully avoiding them as obstacles. When moving the cup towards the trash, it was successfully held in a horizontal position in path segments generated by OpenRAVE (cf. Fig. 4).

Problems can occur if new objects are placed very close to those being manipulated during demonstration. In such cases, the first position of a segment with a high variance and thus the starting position for the classical path plan done by OpenRAVE can already be a position where the robot arm collides with an object. Hence, OpenRAVE will not be able to find a path. Nevertheless, this is still a correct result. In such a case, it is simply not possible to reproduce the skill without a collision in the manner how it was demonstrated. Then, this is not a problem the low-level framework should deal with. A consequence might be to delegate the problem to a higher, symbolic planning level, which, for example, could decide that the obstacle should be moved aside first.



(a) (b) (c)

Fig. 4: Reproduction of the *grasping and emptying* task. First with the known objects only (a), then with an unknown obstacle (one trajectory split up into images b and c). The paths generated by GP (red) and by OpenRAVE (green) are marked. The situation in (a) can be solved by using only GP. The new situation with the obstacle can only be solved by the extended new framework.

Naturally, there is still room for improvement. For example, the system could determine automatically, whether the end effector angle constraints should be handed over to OpenRAVE, as it is now done for every case.

In further experiments, it has to be investigated whether the technique to cut out low constrained regions is applicable to a wide range of problem classes. In these segments handed over to OpenRAVE there are still low variance regions regarding single DOFs. Usually, segments being low constrained for a single DOF only result from a lack of variation during demonstration. Nevertheless, maybe there are some problem classes where this does not apply to.

Another open task is the interaction with a higher level symbolic planner as already mentioned above or the integration into a classical three-tier robotic architecture.

4 Conclusion

We showed that a GP based LfD framework combined with conventional path planning methods can enhance the generalization capabilities of a robot regarding learned skills. Thereby, strengths from both approaches are exploited. The class of unknown situations the robot might face to apply learned skills was expanded successfully by objects and obstacles which are not known from demonstration.

5 Future Work

Additional benefits could be derived from the usage of a robotics learning framework. Instead of relying on a single learning technique to plan motions, learning results could be improved by combining different methods. This could, for instance, be a combination of Gaussian Processes and Human Trainer Feedback or Reinforcement Learning.

As a toolbox which provides the functionality to switch between multiple learning and planning algorithms and to combine them, the TEACHINGBOX [ESCT09] could be used to reach this goal. Prospectively, the current motion planning algorithm is going to be integrated into the TEACHINGBOX framework. Regarding real-world examples like the Katana robot arm, appropriate use cases will have to be developed in this case.

Acknowledgment

This work was supported by the Collaborative Center of Applied Research on Service Robotics (ZAFH Servicerobotik).

References

[BCDS08] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics*. Springer, 2008. In press.

[Bug03] Guido Bugmann. Challenges in verbal instruction of domestic robots. In *Proceeding of the ASER03, the 1st International Workshop on Advances in Service Robotics*, 2003.

[Cal07] Sylvain Calinon. *Continuous Extraction of Task Constraints in a Robot Programming by Demonstration Framework*. PhD thesis, Learning Algorithms and Systems Laboratory (LASA), Ecole Polytechnique Federale de Lausanne (EPFL), 2007.

[CB08] Sylvain Calinon and Aude Billard. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, September 2008.

[DK08] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, July 2008.

[EK08] Staffan Ekvall and Danica Kragic. Robot learning from demonstration: A task-level planning approach. *International Journal on Advanced Robotics Systems*, 5(3):223–234, 2008.

[ESCT09] W. Ertel, M. Schneider, R. Cubek, and M. Tokic. The Teaching-Box: A Universal Robot Learning Framework. In *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009)*, Munich, 2009. `http://www.teachingbox.org` [online; accessed February 25, 2010].

[EZRD02] M. Ehrenmann, R. Zoellner, O. Rogalla, and R. Dillmann. Programming service tasks in household environments by human demonstration. In *IEEE Intl. Workshop on Robot and Human Interactive Communication*, pages 460–467, 2002.

[GWB97] Paul W. Goldberg, Christopher K. I. Williams, and Christopher M. Bishop. Regression with input-dependent noise: A gaussian process treatment. In *Neural Information Processing Systems*. The MIT Press, 1997.

[KMP08] Jens Kober, Betty J. Mohler, and Jan Peters. Learning perceptual coupling for motor primitives. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 834–839. IEEE, 2008.

[KPPB07] Kristian Kersting, Christian Plagemann, Patrick Pfaff, and Wolfram Burgard. Most likely heteroscedastic gaussian process regression. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvalis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 393–400, 2007.

[KSL00] James J. Kuffner, Jr. Steven, and M. Lavalle. RRT-connect: An efficient approach to single-query path planning, November 06 2000.

[Lie77] Alain Liegois. Automatic supervisory control of the configuration and behavior of multibody mechanisms. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-7(12):868–871, 1977.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 1999.

[PS07] Jan Peters and Stefan Schaal. Policy learning for motor skills. In Masumi Ishikawa, Kenji Doya, Hiroyuki Miyamoto, and Takeshi Yamakawa, editors, *ICONIP (2)*, volume 4985 of *Lecture Notes in Computer Science*, pages 233–242. Springer, 2007.

[RW06] Carl Edward Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.