Proceedings of 2004 IEEE/RSJ International Conference on
Intelligent Robots and Systems
September 28 - October 2, 2004, Sendai, Japan

Learning from Demonstration: Repetitive Movements for Autonomous Service Robotics

Holger Urbanek
holger.urbanek@dlr.de

Alin Albu-Schäffer
alin.albu-schaeffer@dlr.de

Patrick van der Smagt
smagt@dlr.de

German Aerospace Center, Institute of Robotics and Mechatronics
P.O. Box 1116, D-82230 Wessling, Germany

***Abstract*— This paper presents a method for learning and generating rhythmic movement patterns based on a simple central oscillator. It can be used to generate cyclic movements for a robot system which has to solve complex tasks. The system is laid out in such a way that multiple motion dimensions, or degrees of freedom of the robot, are represented independent of each other; therefore, an extension to higher-dimensional problems is easily possible. Guiding the robot by holding its end-effector, the user teaches simple movement primitives forming the basis for a more complex task. Each movement primitive is represented in the system using an oscillator combined with a learned nonlinear mapping. These primitives are then optimally combined to a complete solution to the posed problem. Said optimality is obtained using simulated annealing with the A* global search algorithm. Our approach is demonstrated on the problem of wiping a table, but can be used for many typical problems in service and household robotics.**

I. Introduction

While most present-day applications of robots are still restricted to industrial environments, the field of service robotics has a huge potential for growth. The development of lightweight arms and hands as well as the progress in robot navigation systems (mobile platforms) on one side, and the development of advanced humanoid robots on the other, provide the hardware premises for a break-through in this field. Increased sensor feedback capabilities (force-torque, visual, tactile, etc.), and the growth of computing power give the possibility to react more flexibly on changing environments. But providing the robots with an appropriate degree of autonomy which makes them able to use these basic reactive features, as well as providing a simple and efficient human user interface, are still very challenging research topics. Especially the robot programming task, done by the—often technically unskilled—user, has to be as simple as possible. In this concept, programming by demonstration is a widely accepted paradigm. The usual approach, in which the human demonstrates a complete task and the recorded trajectory is followed by the robot, would certainly lack the flexibility needed to survive in common household environments. These environments have a continuously changing, complex structure.

A possible solution is the use of small motion primitives which the user teaches to the robot. Equipped with the appropriate algorithms, the robot uses these primitives in combination with sensory input to accomplish a family of complex tasks. In this paper, we demonstrate such algorithms which may be generally useful in this context. We demonstrate our approach in an application which consists of wiping surfaces which are previously specified by a user or detected by a vision system. Only a "wiping style" has to be taught by the user, in the form of a primitive movement pattern.

The solution presented in this paper is focused on the following topics:

- teaching and learning phase of the primitive movement using a torque-controlled manipulator;
- providing a trajectory generator (as an alternative to usual trajectory interpolation), which can combine these primitive movements to a smooth trajectory and which reacts adequately to external disturbances. Such disturbances could be obstacles or humans which may interact with the robot during execution;
- developing an algorithm which allows automatic movement generation from the taught primitives;
- execution of the entire task in a realistic environment, using the same manipulator, in order to validate the approach and test the required robustness and reactivity.

Most of the previous work on this topic tries to solve only small portions of the overall problem. In [3] the focus is on the combination of primitive tasks, yet no learning or autonomous problem-solving is available. Work on learning focuses mainly on learning a subtask or on how to decide which part of the demonstrated information is relevant or how to construct a neural oscillator [4], [6], [14], but no autonomous task-solving has been applied. [8]–[10] Present a more complete solution for learning a point-to-point or repetitive movement and generating it in a very flexible way. However, primitive movements are not combined in order to solve complex tasks. In [11] a similar system (with neural oscillators) is used to generate biped walking patterns. Another problem of programming by demonstration is presented in [5]—the demonstrated task has to be analysed for its relevant actions and segmented accordingly to ensure a general knowledge of how to solve such a task. Yet this is a quite different problem from the one presented in this paper, where the intention is to combine rhythmic movements to a task more complex than pick-and-place operations. In [2] movement primitives are used to navigate a marble through a Marble Maze. Focus

0-7803-8463-6/04/$20.00 ©2004 IEEE

lies on learning the combination of movement primitives (such as "move marble away from wall") to successfully navigating the marble to its goal. No unforeseen external interaction occurs.

The first part of our paper (section II), which handles learning the movement primitives and the trajectory generation, is based on the results from [8]–[10]. Herein, a novel approach for learning and generation of rhythmic movement patterns is presented. In contrast to this approach, however, we have reduced the complexity while maintaining its flexibility. Learning is achieved by using a sum of weighted Gauss kernels which maps a centralised oscillator signal to the desired movement. To increase the required robustness of execution, the difference between desired and actual robot position is used to slow down the oscillator and hereby delay further generation of the trajectory. Thus the system is able to cope with external disturbances or slows down if the robot cannot follow the trajectory with the desired velocity.

The second part of the paper (section III) presents a method to autonomously combine the learnt movement primitives in order to solve the given task. An algorithm to cover the entire surface by primitive movements is presented in section III-A, while finding an optimal execution order of this movements is addressed in section III-B. Finally, experimental results are presented and discussed.

II. Learning and generating rhythmic movements

In this section we present the structure of the trajectory generator for the entire movement, as well as how the movement primitives are represented in the system.

The trajectory generator should reproduce the recorded trajectory with the desired degree of flexibility and be able to combine these primitives to a smooth movement. Therefore it should have following features:

- it should be possible to stop the generation at any point of movement;
- the learnt movement should be smooth in order to appear more natural;
- to enhance smoothness between transitions from one primitive movement to another, a dynamic interpolator to a moving goal is needed. This is also useful if the robot is displaced by external forces from the trajectory (e.g., by a human pushing it away), and has to return smoothly on the path after the disturbance disappears;
- the speed of the movement must be adjustable (e.g., to cope with dynamic constraints of the robot).

Most of these demands are met by the system developed in [8]–[10]. There, an oscillator that can be slowed down by an input signal is used as a control policy. This control policy allows the generation of the desired movement that is time-invariant, and also allows a rudimentary reaction on disturbances. The oscillator signal is used as a driving signal for the weighted sum of Gauss kernels in order to generate the basic movement, which in turn can be modified (e.g., scaled, translated, or rotated) without disturbing the stability of the whole system. To ensure a smooth trajectory, a first-order filter is used. The difference between the desired and the actual robot position slows down the oscillator, and therefore the generated movement is slowed down in turn.

However, the presented system has some disadvantages. In the following we will describe the system taken from [9] and introduce some our modifications to solve these disadvantages.

A. The oscillator

For determining the progress of the learnt movement primitive, [9] suggests the following nonlinear oscillator:

$$\dot{z} = -\frac{\mu}{E_0}(E - E_0)z - k^2 u, \tag{1}$$

$$\dot{u} = z\left[1 + \alpha_u(\tilde{y} - y)^2\right] - 1, \tag{2}$$

with $E(u,z) = \frac{z^2}{2} + \frac{k^2u^2}{2}$ as the actual energy of the oscillator, E_0 the desired energy, μ the desired convergence rate, and k as the desired frequency. $\tilde{y}$ Denotes the measured and y the ordered robot position. $z, u \in \Re$ Are state variables. α_u Is a scaling factor for the position error feedback. High values for α_u or high position errors $(\tilde{y}-y)$ draw $\dot{u}$ towards zero and therefore stop the oscillator. For $\alpha = 0$, Eqs. (1)–(2) reduce to a second order system with the variable damping $d = \mu(E/E_0 - 1)$. It follows that $d < 0$ inside the limit cycle and $d > 0$ outside, so that the circle $E = E_0$ becomes an attractor. Fig. 1 illustrates the convergence to the limit cycle of the proposed oscillator.

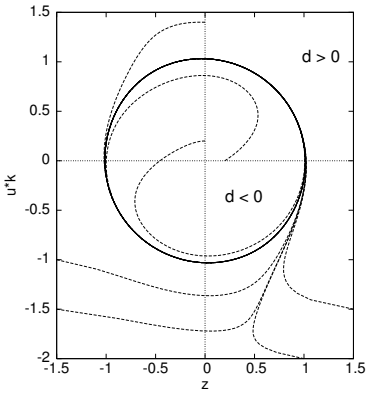

Fig. 1. The oscillator proposed in [9]. It can be clearly seen that the circular limit cycle is an attractor. The plot is drawn for $\tilde{y} - y = 0$, $E_0 = 0.5$, $k = 2\pi$, $\mu = 10$, and $\alpha_u = 200$.

The driving signal ϕ for the learning function in section II-B is generated by using atan2:

$$\phi = \text{atan2}(z, ku). \tag{3}$$

This converts the oscillator states u and z into a toothsaw-shaped signal. However, this oscillator is not able to stop equally distributed over the interval defined by the driving signal. Since only $\dot{u}$ is driven to zero by the position error, the oscillator will continue to move until $\dot{z}$ is also zero.

This means that up to a half rotation may be performed before $\dot{z}$ reaches 0 and ϕ ceases to change.

To enable a immediate stop at any point of the movement, we introduce a simpler oscillator that directly generates a toothsaw-shaped signal:

$$\phi_i' = \phi_i + \frac{m}{1 + (\alpha_\phi |\tilde{f} - \tilde{y}|)^{k_\phi}}, \tag{4}$$

$$\phi_{i+1} = \begin{cases} \phi_i', & \text{if } \phi_i' < 1, \\ \phi_i' - 1, & \text{else}, \end{cases} \tag{5}$$

where $\tilde{f}$ is the desired[1] and $\tilde{y}$ the measured position. α_ϕ And k_ϕ are fine-tuning constants; α_ϕ defines a low reaction zone and $k_\phi > 1$ the order of the braking effect. The slopes of the oscillator signal are in the range $[0, m)$. By design, this simpler oscillator has no problems with slowing down; Fig. 2 shows a direct comparison of the breaking performance of the both oscillators.

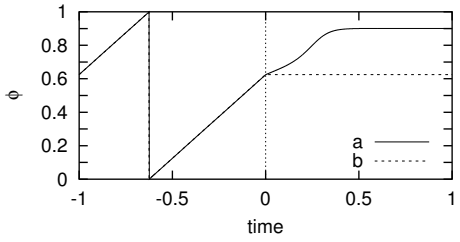

Fig. 2. Breaking performance. Both oscillators (a is the original one as proposed in [9] and b is our) get a complete stopping signal ($|\tilde{f} - \tilde{y}| = \infty$) at time 0. As can be clearly seen, the signal of b halts immediately, while the original oscillator a may need some time to react.

Notice that the convergence property of the oscillator of Eqs. (1) and (2), which is not present in Eqs. (4) and (5), is not truly exploited in [9]. The smooth convergence to the desired trajectory is obtained instead by a linear filter, similar to the one mentioned in section II-B.

B. Learning a primitive movement

We approximate the movement by a sum of Gauss kernels for each degree of freedom independently:

$$f = \sum_{i=1}^{N} w_i \exp\left[-\frac{1}{2}(c_i - h_i \phi)^2\right]. \tag{6}$$

The parameters w_i, c_i, and h_i have to be determined to optimally fit the movement. Many methods exist; we have chosen to use the Fletcher-Reeves-Conjugate-Gradient optimisation methods, since it exhibits excellent results after only few iterations.

The parameterisation of the primitive movements in w_i, c_i, and h_i can, at a later stage, be used to classify such primitives [10].

[1]We have chosen the desired position $\tilde{f}$ instead of the ordered position y, to ensure a trajectory as near as possible to the planned.

C. Finalisation

The function f computed in Eq. (6) describes the generated position, which still has to be transformed (i.e., translated, rotated, and scaled):

$$\tilde{f} = H\ f, \tag{7}$$

with H being a homogeneous transformation matrix. Therefore $\tilde{f}$ denotes the desired position.

To ensure a smooth trajectory, a second-order filter is applied:

$$\ddot{y} + \alpha_y \dot{y} + \beta_y (y - \tilde{f}) = 0, \tag{8}$$

where

$$\alpha_y = \frac{d}{m},\ \beta_y = \frac{k}{m}, \tag{9}$$

with d as damping, m as mass and k as spring constant. By using a filter instead of an extra trajectory interpolator (e.g., Bézier splines) our system includes the possibility to move towards a moving goal, while with an extra interpolator this would be difficult to achieve.

An additional feature of the presented system, which generates rhythmic movements, is the fact that it works for an arbitrary number of degrees of freedom (DOF). The oscillator acts as a centralised control policy to synchronise the different DOFs, while every DOF is handled separately thereafter, by having one approximator (6) per DOF. Our table wiping scenario requires only 2 DOFs, as the region is planar and we are using Cartesian impedance control for the robot.

Fig. 3 shows the structure for multiple DOFs.

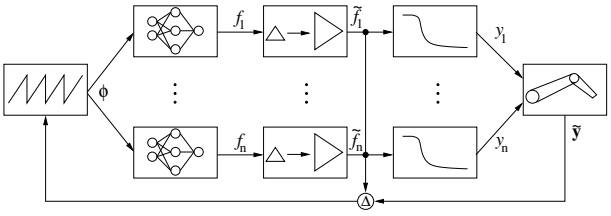

Fig. 3. How to enhance the presented system to multiple DOFs.

Our system, adapted from [8]–[10], demonstrates an exceptional robustness. The generated movement can be disturbed in any way, which results in the oscillator stopping for the duration of the disturbance—without losing significant parts of the desired movement. Even if the trajectory cannot be followed by the physical robot (e.g., the robot is too slow or there are obstacles in its way), our system adopts quickly to this constraint and changes the path-generation accordingly.

III. Building a complex task from learnt movement primitives

In the previous section we have introduced a system to learn and represent primitive movements which are taught by demonstration. We will now show how we can combine such primitives to autonomously construct a complex movement, e.g., in order to wipe a complete table. Contrary to approaches where the whole task is taught, our two-stage

approach allows us to solve various different tasks without requiring further operator interaction. For instance, we can wipe any area of the table, as dirt is detected by a camera, or clean windows or arbitrary surfaces.

The presented system has been implemented and tested on the Light-Weight-Robot II (LWR-II), a 7-DOF robot arm developed at the Institute of Robotics and Mechatronics of the German Aerospace Centre (DLR). In this particular approach, the LWR-II runs with Cartesian impedance-control [1], which allows us to interact with the robot arm by grasping and holding it during motion. In the teaching phase, the Cartesian stiffness for translational directions is set to zero, while the orientations are kept stiff. During the execution phase, all Cartesian stiffness values are high, except for the normal direction to the surface. This value is zero and the robot is commanded to exert a constant desired force in this normal direction. Since the impedance controller is implemented based on measurements of joint torque sensors, every collision along the structure of the robot, not only at the tip, is detected and slows down or even stops the movement.

In order to solve the given task, we define the following sub-tasks to wipe a table from recorded motion primitives:

A fill the wiping region with primitives, according to several optimality constraints;
B compute an optimal path;
C execute the plan.

A. Filling the wiping region

In order to wipe the whole table, the primitive wipe operation must be distributed over the area with as little overlap as possible, while not leaving out any part of the wipeable region. This planning problem has some similarities to cutting or packing problems (e.g., [12]). The fundamental difference, however, is that in our case minimal overlap is allowed (and even required), whereas leaving out parts is strongly penalized.

For reasons of simplicity we assume that the whole area that is described by the convex hull around the primitive wipe operation is also covered by that operation. Thus this region can be exhaustively described by the polygon described by it. The whole table can then be considered to be wiped when the primitive wipe polygons are optimally distributed on the table.

Finding the optimal distribution of primitive wipe polygons on the table is an NP-problem, meaning that finding a solution to this problem grows non-polynomially with the number of polygons. Rather than following this path, in order to find a solution in a reasonable amount of time we follow the following heuristic. We rasterise the wipeable region in, e.g., 100×100 segments or *pixels*. The primitive wiping polygons are drawn on this region using standard drawing algorithms (c.q. the Bresenham line draw and line scan fill), while it is counted in $p_{i,j}$ how often a segment at location (i, j) is drawn into. We can thus define the following error:

$$\text{err}_{i,j} = \begin{cases} d_1, & \text{if } p_{i,j} = 0 \text{ and } R \\ d_2(p_{i,j} - 1), & \text{if } p_{i,j} > 0 \text{ and } R \\ d_3 \cdot p_{i,j}, & \text{if } \bar{R} \end{cases} \quad (10)$$

with R denoting the inside of the wiping region and $\bar{R}$ its outside. We have empirically chosen the constants $d_1 = 20$, $d_2 = 1$, and $d_3 = 5$. The sum over all these per pixel-error values defines the following global error function:

$$\text{ERR} = \sum_{i,j} \text{err}_{i,j}. \quad (11)$$

A perfectly filled region has the error value of ERR $= 0$, and the value of ERR increases with more overlap and unfilled areas. The cost to compute ERR is $O(m\ n^2)$, with m being the number of wiping movements and n the segmentation in each of the two possible directions.

Every primitive wipe polygon is determined by its position (x, y) in the wipe region, its rotation angle, and its scaling factor (which was typically chosen between 0.9 and 1.3). A further free parameter of the system is the number N of polygons used.

We use simulated annealing with a linear cooling scheme to find an optimal solution within the $4N + 1$ parameters. The quality of the resultant solution depends on the fineness of rasterisation as well as the cooling scheme; the slower the system is cooled, the lower the energy of the found solution is expected to be. Both factors are limited by the available computing time; a wiping robot taking longer than a few minutes to compute its solution is hardly useful.

B. Optimising the movement plan

To ensure an acceptable wiping-time and a naturally looking wiping movement, we use a heuristic to search a short path over the centres of the wiping-movements. This is similar to the Traveling-Salesman problem, and many viable algorithms exist. Since this particular task is not very complex and the number of polygons in a wipe region is limited, we decided to use the A*-algorithm.

A* (e.g., [13]) is a modified breadth-first search (BFS) that utilizes a heuristic to estimate the distance from the current to the goal state. Using a greedy approach, only the shortest path is expanded at any time. As long as the heuristic underestimates the real distances, A* is guaranteed to find the shortest path, as BFS would.

For the heuristic h we choose:

$$h = \gamma\ |O|, \quad (12)$$

with $|O|$ as the number of open nodes and γ as a distance estimator that has to be set accordingly. One problem with A* is that it requires a large amount of memory (up to $O(N!)$ for a badly chosen $\gamma \approx 0$, where N denotes the number of nodes), but overestimating the distance (and hence turning A* into an A) results into a sufficient short path while memory usage is cut down significantly.

Future work will include an optimisation over the starting and end points of the movement primitives, perhaps even the directions of the starting and end points will be

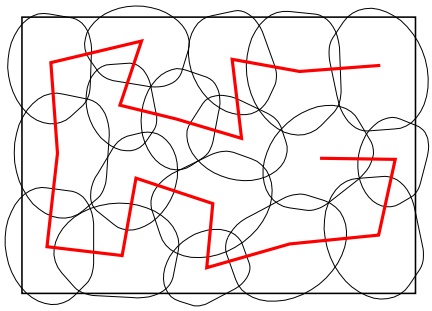

Fig. 4. Example for a short path over the centres of the primitive wiping movements (displayed by their convex hull).

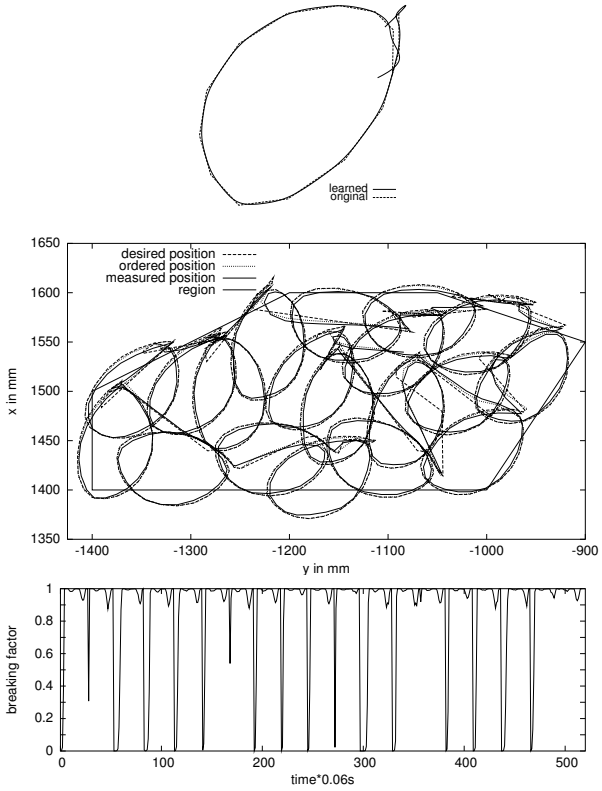

Fig. 5. Recorded movements (middle) from an example run with roundish movements (top). Especially interesting is the breaking factor (see Eq. (4)) as the spikes are indicating the switching to a new transformation Matrix H. It is evident that the roundish movement primitive can be easily followed by the robot.

included in the evaluation, to achieve an even more natural impression.

C. Executing the movement plan

The filling (see section III-A) and the sequencing (see section III-B) together define the *wiping plan* which, in turn, is executed by the system described above.

The execution of the movement plan is straightforward. Since the repetitive generation of the primitive movement is generated by the system described in section II, only a new movement transformation matrix H has to be used in Eq. (7) whenever the oscillator (Eqs. (4) and (5)) takes its backward step. Smoothness of the trajectory is still guaranteed by the second-order filter. Additionally the maximal velocity is limited. The positional error $(\alpha_\phi|\tilde{f}-\tilde{y}|)^{k_\phi}$ (see Eq. (4)) stops the oscillator whenever external influences cause the robot to do so, so that no significant part of the next movement primitive is lost. It can be clearly seen from Fig. 5 that the breaking factor slows down the trajectory generation whenever a new transformation Matrix H is applied.

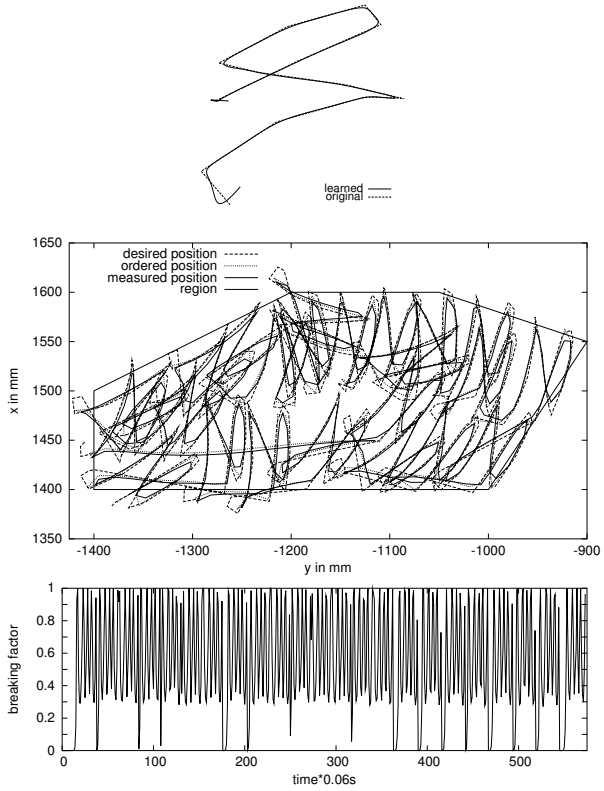

Fig. 6. Recorded movements (middle) from an example run with zigzag movements (top). In contrast to the roundish movement (see Fig. 5) the breaking factor indicates that the impedance controlled robot has quite a problem to follow the zigzag movement with the given speed. Yet the wiping plan is executed satisfactorily.

The roundish movement is no problem for the impedance controlled robot to follow—in contrast to the zigzag-movement shown in Fig. 6. Although there the oscillator is almost always slowed down, the actual trajectory of the robot is still similar to the desired one, so the wiping action is accurately executed.

To demonstrate a momentary external disturbance, the LWR-II was simply gripped and moved away from its ordered position while executing a movement plan similar to Fig. 5. Fig. 7 presents a part of the movement recordings. As demanded, the further generation of the trajectory ceases as long as the disturbance lasts.

IV. Future Work

Further improvements would include an evaluation of genetic algorithms (GA) for optimising the wiping layout, since GAs should be able to evaluate $O(n^3)$ layouts per evaluation-step (n denotes the size of a generation, see [7])

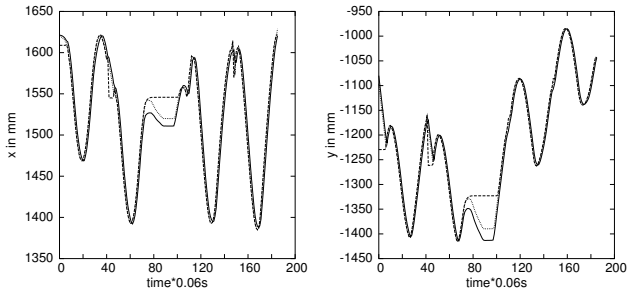

Fig. 7. x and y recordings of a roundish movement with a momentary disturbance around time 80 – 100. Clearly you can see how the further trajectory-generation ceases for the duration of the disturbance.

in contrast to Simulated Annealing which can evaluate only one per step.

Furthermore different path optimising algorithms will be evaluated, especially randomised ones, as they allow suboptimal solutions to be generated. In a demonstration scenario computing time is always short, so it is more important to find a quite good, yet suboptimal solution than waiting for ever for the perfect solution. Randomised Algorithms (such as GA) are able to deliver this—like our A* by decreasing γ. Also path optimisation should also consider the start- and end-points of the wiping primitive and preferably also the directions of these. Randomised algorithms should also be able to cope with this matter.

Another interesting aspect would be the extraction of one primitive wiping movement out of a continuously demonstrated wiping action, as we (humans) are better able to generate a sequence of rhythmic movements rather than just exactly one—so this is an ergonomic issue.

Eventually the robot will be enhanced to 10 DOFs (LWR-II on a moving platform) so the area it can reach will be greatly enhanced. Furthermore, the system must be enhanced such that non-wipe zones are taking into account in the planning phase.

V. Conclusion

An approach for teaching and solving complex tasks in a service robotics scenario has been proposed. The human teacher demonstrates simple movement primitives by guiding the robot by the hand. The system first learns these movement primitives, then combines them autonomously to a complex movement plan. The idea is exemplified in an application in which the robot has to wipe a surface which is specified online. A major advantage of the proposed approach is the ability to cope with temporary disturbances and to adapt to robot limitations.

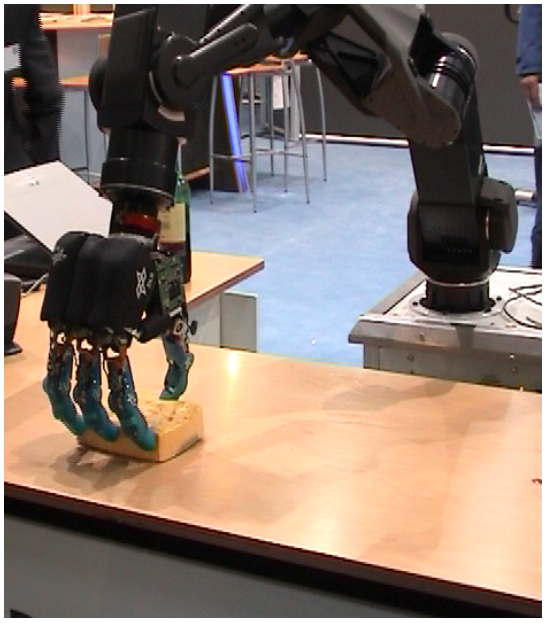

Fig. 8. The wiping LWR-II robot.

References

[1] Albu-Schäffer, Alin and Hirzinger, Gerd. Cartesian impedance control techniques for torque controlled light-weight robots. *ICRA*, pages 657–663, 2002.

[2] Bentivegna, Darrin C., Atkeson, Christopher G., and Cheng, Gordon. Learning from observation and practice using primitives. *Workshop on Robot Programming by Demonstration, IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[3] Brunner, Bernhard, Vogel, Jörg, and Hirzinger, Gerd. Aufgabenorientierte Fernprogrammierung von Robotern. *at - Automatisierungstechnik*, 49(7):312–319, 2001.

[4] Doya, K. and Yoshizwawa. Adaptive syncronization of neural and physical oscillators. *Advances in Neural Information Processing Systems*, 1992.

[5] Ehrenmann, M., Zöllner, R., Rogalla, O., and Dillmann, R. Programming service tasks in household environments by human demonstration. *11th IEEE Int. Workshop on Robot and Human interactive Communication (ROMAN)*, pages 460–467, 2002.

[6] Ermentrout B., and Kopell, N. Learning of phase lags in coupled neural oscillators. *Neural Computation*, 6:225–241, 1994.

[7] Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts et al., 1989.

[8] Ijspeert, Auke Jan, Nakanishi, Jun, and Schaal, Stefan. Learning attractor landscapes for learning motor primitives. *Advances in Neural Information Processing Systems*, 15, 2002.

[9] Ijspeert, Auke Jan, Nakanishi, Jun, and Schaal, Stefan. Learning rhythmic movements by demonstration using nonlinear oscillators. *Proceedings of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, pages 958–963, 2002.

[10] Ijspeert, Auke Jan, Nakanishi, Jun, and Schaal, Stefan. Movement imitation with nonlinear dynamical systems in humanoid robots. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398–1403, 2002.

[11] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion with dynamical movement primitives. *Workshop on Robot Programming by Demonstration, IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[12] Milenkovic, Victor J. Rotational polygon overlap minimization. *13th Annual Symposium on Computational Geometry (SoCG)*, 1997.

[13] Nilsson, Nils J. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, San Francisco, reprint edition, 1986.

[14] Nishii, J. A learning model for oscillatory networks. *Neural Networks*, 11:249–257, 1998.