Reactive Phase and Task Space Adaptation for Robust Motion Execution

Peter Englert and Marc Toussaint

***Abstract*— An essential aspect for making robots succeed in real-word environments is to give them the ability to robustly perform motions in continuously changing situations. Classical motion planning methods usually create plans for static environments. The direct execution of such plans in dynamic environments often becomes problematic. We present an approach that adapts motion plans by feeding changes of the environment into a transformation of the plan in task space. Furthermore, the progress in the plan is defined with a phase variable that is updated adaptively according to the actual task progress. This phase variable releases the strict time compliance that many motion planning methods bring along. The main benefit of our approach is the ability to do this adaptation in a computational efficient manner during the execution of the motion. Thus, the gap between the motion planning and motion execution stage is bridged by continuously transforming geometric and dynamic features of a reference plan to the current situation. We evaluate the performance of our approach by comparing it to alternative methods such as dynamic motion primitives and continuous replanning on several simulated benchmark tasks. Moreover, we demonstrate the real robot applicability on a PR2 robot platform.**

I. INTRODUCTION

There are many different ways of designing motion plans for robots. The most widely used techniques include trajectory optimization [1], human demonstration [2] and the manual programming of movements [3]. In most cases, the outcome of such techniques is a time-dependent trajectory, which describes a motion of a robot that fulfills a task in a static scenario. This trajectory is then usually passed on to a controller that executes it. In scenarios where an exact model of the environment exists and no unforeseen events happen, such an approach is usually sufficient. An example for such scenarios are many manufacturing tasks, where the robot just has to execute the exact same motion in the same fixed environment over and over again.

However, in real-world scenarios such an approach is often unsuitable. One reason is that real-world scenarios are dynamically changing (e.g., humans or objects move). Another reason is that some information may often not be available during the planning phase (e.g., the exact object position can only be measured accurately when the robot is in a certain range to it). In such scenarios, the approach of planning and subsequent tracking of trajectories often would lead to task failure.

To make robots also be able to succeed tasks in such real-world scenarios, it is necessary to give them the ability to react and adapt on changes. Therefore, methods are required that can include state changes in a feedback-driven way into the motion execution. Planning behavior that is reactive and adaptive to continuous changes is non-trivial. One problem is that the definition of a process of how the robot should react to certain changes is more difficult than in static scenarios. Especially if there are many changes or changes that cannot be predicted well. Another problem is how such behavior should be represented in a form that it is executable.

Peter Englert and Marc Toussaint are with the Machine Learning and Robotics Lab, Universität Stuttgart, Germany
peter.englert@ipvs.uni-stuttgart.de
marc.toussaint@ipvs.uni-stuttgart.de

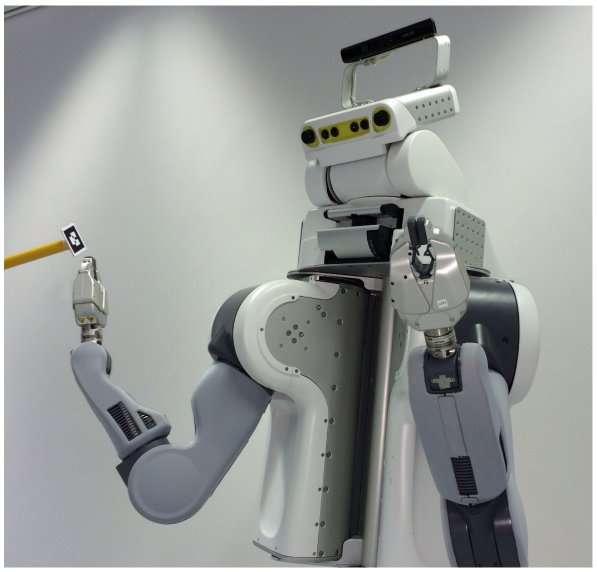


Fig. 1. The Willow Garage PR2 robot moves with its right hand towards the goal state, which is specified with the AR marker on the rod. The goal state is moved during motion execution and the robot continuously adapts a reference plan to the current situation.

Instead of already taking changes in the planning phase into account, we present in this paper an approach that is based on creating the motion in static scenarios and generalizes to state changes by adapting the plan continuously during execution. Therefore it is still able to rely on classical motion planning methods such as optimization or imitation for task definition. Additionally, it can react during the plan execution to occurring state changes. Thereby, we use the outcome of such motion planners more as a guidance for achieving task success and not as a trajectory that should be tracked exactly.

The adaptive behavior of our approach is achieved in two fashions. First, state changes and online perception feed into a transformation of geometric and dynamic features of the reference plan in task space. Thereby, the shape of the reference plan is adapted to the new situation and it is guaranteed that the robot and the goal state are always located on the current plan. Second, a phase variable is used to measure task progress. This phase variable is responsible for connecting the current robot position and the corresponding position in the reference plan. It guides the movement towards reaching the goal and is adaptively increased depending on the actual progress and state changes. Our approach is a compromise between fully new planning and myopic trajectory following control. It is computational efficient and provides smooth transformations of the plan to the current situation.

Throughout this paper, we make two assumptions on the task that need to be fulfilled, so that our approach can be applied: (1) Reaching the final state is the main goal for achieving task success. (2) Exact time compliance is not essential for achieving task success. The first assumption is not mandatory and a reformulation may also be eligible for different variants (e.g., intermediate goals). The second assumption is more strict, due to changes of the environment an exact time compliance with respect to a reference plan is very difficult to achieve. Both assumptions restrict the class of possible tasks. Nevertheless, there are still many tasks where the exact time execution is not essential and the generalization abilities that come along with our approach are more beneficial. For example, at grasping tasks it is usually irrelevant if the execution time is 13 or 16 seconds and it is more important that the object can be grasped reliably and at different positions.

The rest of the paper is structured as follows: In Section II we present first an overview of related work on different motion adaptation approaches for dynamic scenarios. Afterwards, we present in Section III our approach that is based on task space transformation as well as phase space adaptation and provide a concrete algorithmic implementation. Then, we evaluate in Section IV our approach in multiple simulated benchmark tasks and compare it to alternative methods. Furthermore, we demonstrate the real robot applicability on a Willow Garage PR2 (see Figure 1) by continuously adapting a reference motion to the current goal state, which changes during motion execution.

II. Related Work of Motion Adaptation Methods

In the following section, we present an overview of different motion adaptation approaches. The main focus will be on the adaptation technique and how the time handling is done during the adaptation.

First, we give an overview of online replanning approaches that adapt motions by continuously replanning again during execution. Afterwards, we present different phase space adaptation methods that avoid the strict time dependence by substituting the time with a phase variable. Finally, we review stationary controller approaches, where the next action is chosen by purely depending on the current state.

A. Online Replanning

One of the most intuitive approaches is to continuously replan the motion again during execution. For this, classical motion planning methods are adapted in such a way that they can be applied continuously during execution [4]. Rapidly-exploring random trees have been modified to dynamic scenarios by just repairing the invalid parts of the tree that are affected from the dynamic changes [5]. Elastic strips are an approach for reactive motion executing by adapting planned motions to changes in dynamic scenarios [6]. They are based on incremental modification of the planned motion with energy functions.

Another approach on the control level is model predictive control [7], which iteratively optimizes a cost function for a finite horizon into the future. It has been used in different areas of robotics like tracking mobile robots [8] or generating walking gaits for humanoid robots [9].

Using online replanning methods has many advantages since it extends well-known offline planning methods to dynamic scenarios. Depending on the planning method it may often even be possible to find completely new plans if the previous plan is not executable anymore (e.g., because an obstacle blocks the path). However, using online replanning in real-world scenarios is not straightforward. Depending on the planning algorithm and the complexity of the current scene, the computational costs often restrict the online replanning to a low frequency. Since most optimization algorithms need multiple iterations for convergence, it is difficult to specify an upper bound for the required computational time until the algorithm converges. Therefore, it is difficult to integrate them into a real-time motion execution framework with fixed computational costs. Another drawback for using classical planning methods in dynamic scenarios is that many of them are still time-dependent. For example, most trajectory optimization methods require a definition of the motion duration, which is difficult to specify when changes in the environment occur.

B. Phase Space Adaptation

Phase space adaptation methods use a phase variable s to release the strict time dependency of motion plans. This phase variable is a substitute for time that controls the progress in the plan during execution. It usually has a predefined start and end value, which represent the first and last value of the motion plan. Each intermediate value corresponds to exactly one entry of the plan and maps the current robot position to a corresponding position in the plan. The advantage of a phase variable is that the time evolution of the phase can be modified easily.

The previous work that is most related to our approach are movement primitives, which are parametrized representations of movements. The main purposes of movement primitives are to have a compact representation of the movement, to be able to generalize the motion and to provide a way for generating control commands. Dynamic movement primitives [10] are a widely used type of movement primitives. One reason for that is their representation, which can be easily adapted to new situations (e.g., time-scaling, goal change). Another reason is that they are suitable for learning algorithms. For example, they can be created via reinforcement learning [11] and imitation learning [12]. Dynamic movement primitives are based on a linear spring-damper system that is combined with a nonlinear forcing function. The forcing function is responsible for specifying the shape of the movement and vanishes over time. Probabilistic movement primitives [13] are a recent probabilistic approach to movement primitives. They represent movements as probability distribution over trajectories and achieve useful properties such as blending and parallelizing of movements by using probabilistic operations. There exist different techniques of how to propagate the phase variable in movement primitives. In the classical form of dynamic movement primitives [10], the phase variable s is initialized with 1 and is updated with

$$\tau \dot{s} = -\alpha s \tag{1}$$

in each time step until it reaches the value 0. Here, τ is a scaling constant to make the movement slower or faster and α is a predefined constant. An alternative update rule [14] for the phase variable is

$$\tau \dot{s} = \alpha s \frac{1}{1 + \beta (\boldsymbol{y}^* - \boldsymbol{y})^\intercal (\boldsymbol{y}^* - \boldsymbol{y})}, \tag{2}$$

where β is a constant, $\boldsymbol{y}$ the current robot state and $\boldsymbol{y}^*$ the desired state. With this update rule the progress is slowed down when the current state $\boldsymbol{y}$ is far away from the desired state $\boldsymbol{y}^*$. We will propose a phase adaptation mechanism that monitors the actual progress in task space during execution, and compare this to DMPs in our expriments.

Phase variables are widely used in robot walking motions [15], [16]. In [15] a time-scaling control law is introduced. The time is transformed by a constant-time scaling law

$$s = \lambda t \tag{3}$$

with a constant λ. A cubic polynomial time scaling function is derived that leads the system from one constant-time scale λ_1 to another constant-time scale λ_2. This constant time-scaling approach has been applied on a biped robot to speed up and slow down the walking motion.

In physics-based animations a phase variable has been used for executing rotational movements (e.g., cartwheel, flip) with animated characters [17]. They use the phase variable to position the character in the revolution state. As phase variable an angle between different body parts is used. The phase variable is also used to divide the movement in different stages. For each stage a different controller is designed.

In [18] they design a distance indexed reference trajectory for a helicopter robot. The reference trajectory is parametrized with the distance to the goal state. This reference trajectory is used in an outer loop of a cascade controller to provide new targets for a low level position controller.

While these classical phase adaptation schemes usually focus on cyclic movement we propose methods to adapt the execution phase of a plan depending on feedback on task progress and change in goal location.

C. Stationary Controller

Stationary controllers are an alternative to the previously mentioned approaches, since they provide a time-independent mapping of the current state to a desired control command. They have been widely used in reinforcement learning algorithms. Their behavior depends just on the state, which is in many tasks a more reasonable decision measure for choosing the next action than absolute time (e.g., a grasping task should not depend mainly on time, but more on the state of the grasp, if it is stable or not). However, a stationary controller for complex motions usually needs to be a non-linear since the expressiveness of linear controller is very limited. An example of a nonlinear representation of state-feedback controller are radial basis function networks. In [19] they are trained in a reinforcement learning setup by simulating the system to learn a controller that maximizes a reward function. An alternative to defining a reward function was presented in [20] by learning the controller from human demonstrations. Generalization can be achieved in such controllers by also taking external states (e.g., goal position) as an input to the controller function [21]. A drawback here is that especially for high-dimensional state spaces the training time with gradient descent methods becomes computational expensive.

Another approach of time-independent control is space-indexed dynamic programming [22]. There, a spatial index variable is used to measure how far the robot moved along the reference trajectory. Based on this spatial index, they present space-indexed versions of the differential dynamic programming and policy search by dynamic programming algorithms. An approach based on learning from demonstration is presented in [23], [24], which learns a time-independent dynamic model of a set of movements in task space. They learn a hidden Markov model to encode spatial and temporal information of the demonstrated motions. Gaussian mixture regression is used for reproducing the motion, which gets the current position as input and computes a desired velocity.

The parametrization over states instead over time is very intuitive since the controller always chooses an action based on the current state. Therefore, when changes in the state occur they are directly fed as input into the controller. Drawbacks are the high computational training costs in higher dimensional state spaces and the difficulty to verify robust behavior of the nonlinear controller for all different states.

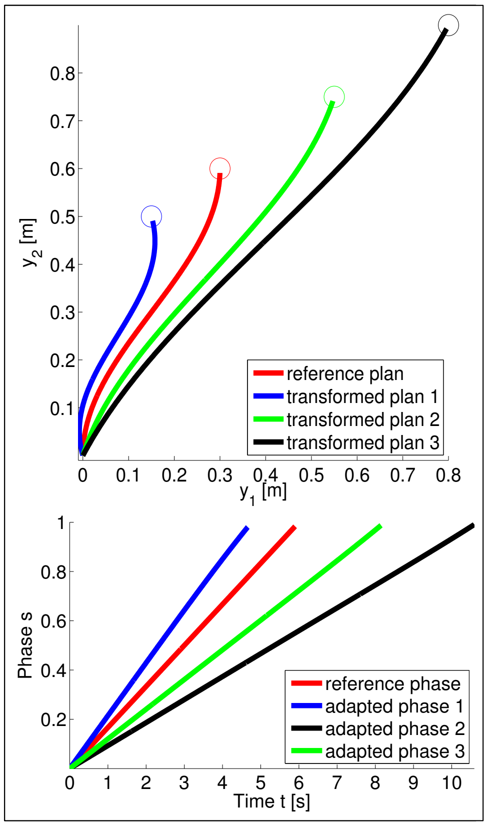
III. Reactive Phase and Task Space Adaptation for Robust Motion Execution

In the following section, we describe the technical details of our approach. First, we present the notation and some background on how to execute motions in task spaces. Then, we describe the strategies used for the task space transformation and the phase space adaptation. Finally, we present a concrete algorithmic implementation which combines all these components.

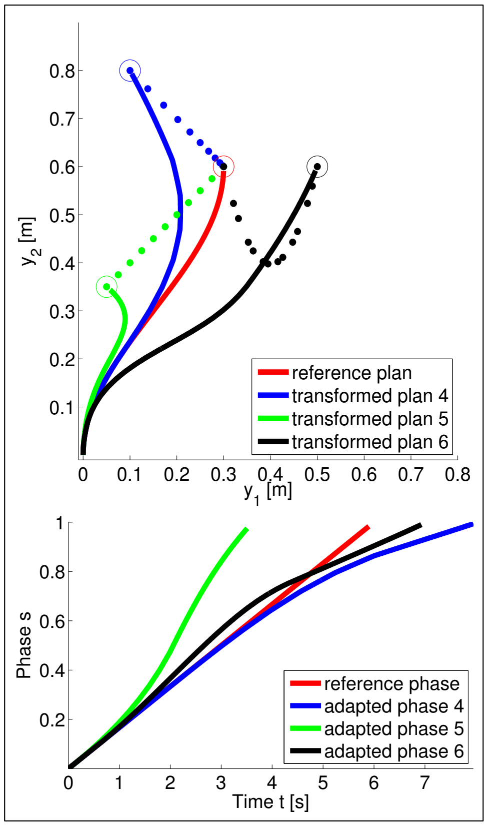
A. Notation and Background on Motion Execution

Throughout this paper, we use the following notation:

- Joint state $\boldsymbol{q} \in \mathbb{R}^n$
- Task state $\boldsymbol{y} = \phi(\boldsymbol{q}) \in \mathbb{R}^m$
- Task goal state $\boldsymbol{g} \in \mathbb{R}^m$
- Action command $\boldsymbol{u} \in \mathbb{R}^k$



(a)



(b)

Fig. 2. The graphs show example motions of our approach. The top graphs shows task space transformation (see Section III-C) and the bottom graphs show the corresponding phase adaptation (see Section III-B). The graphs with the same color in top and bottom graph correspond to the same motion. The red graph shows in all graphs the reference plan $\boldsymbol{\tau}^{\text{ref}}$. The adapted plans 1–3 show different executed plans $\boldsymbol{\tau}^{\text{plan}}$. In the left box, the goal position has an offset to the reference position before the execution starts. On the plots on the right side the goal position moved during the execution. Here, the colored dots represent the movement of the different goal positions during execution.

- Euclidean norm $\|\cdot\|$
- Trajectory in task space $\boldsymbol{\tau} = \{\boldsymbol{y}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_T\}$

Although trajectories are usually represented as a finite sequence of states, we assume in the following that we can also evaluate them like functions $\boldsymbol{\tau}(t)$ for $t \in [0, T]$ by interpolating them via quadratic B-splines.

In general, we assume that the input of our approach is a reference plan $\boldsymbol{\tau}^{\text{ref}}$ in task space. Our approach adapts this reference plan to changes of the environment during execution. Representing motions in arbitrary task spaces has many advantages. The design of motions with imitation techniques (e.g., motion capture) or cost functions (e.g., distance to obstacles) is easier in task spaces than in the configuration space of the robot. Some task spaces are also robot independent (e.g., cartesian world coordinates), which allows the transfer of task knowledge between different robots. For executing motions that are defined in task spaces a method is needed that computes the corresponding action commands of the robot. Operational space control [25] is a control method for executing motions in task space. Thereby, it computes the control command $\boldsymbol{u}$ by minimizing at each time step the optimization problem

$$\boldsymbol{u}^\star = \arg\min_{\boldsymbol{u}} \|\boldsymbol{u}\|_H^2 + \|\boldsymbol{\Phi}(\boldsymbol{q})\|^2 \tag{4}$$

with a task vector

$$\boldsymbol{\Phi}(\boldsymbol{q}) = \begin{pmatrix} \sqrt{c_{\text{pos}}}(\ddot{\phi}_{\text{pos}}(\boldsymbol{q}) - \ddot{\boldsymbol{y}}^\star_{\text{pos}}) \\ \sqrt{c_{\text{dir}}}(\ddot{\phi}_{\text{dir}}(\boldsymbol{q}) - \ddot{\boldsymbol{y}}^\star_{\text{dir}}) \\ \sqrt{c_{\text{col}}}(\ddot{\phi}_{\text{col}}(\boldsymbol{q}) - \ddot{\boldsymbol{y}}^\star_{\text{col}}) \\ \vdots \end{pmatrix}. \tag{5}$$

Here, $\ddot{\boldsymbol{y}}^\star$ are desired accelerations in the various task spaces. In our case, these desired accelerations are defined by specifying a desired PD behavior within the task space,

$$\ddot{\boldsymbol{y}}^\star = K_p\ (\boldsymbol{y}^\star - \phi(\boldsymbol{q})) + K_d\ (\dot{\boldsymbol{y}}^\star - \dot{\phi}(\boldsymbol{q}))\ . \tag{6}$$

A wide range of different task maps is possible. For example, in Equation (5) the first line describes the distance between the endeffector $\phi_{\text{pos}}(\boldsymbol{q})$ and the target $\boldsymbol{y}^\star_{\text{pos}}$. The second line represents the distance between the orientation of the endeffector $\phi_{\text{dir}}(\boldsymbol{q})$ and the target orientation $\boldsymbol{y}^\star_{\text{dir}}$. The third task map is responsible for collision avoidance by incorporating the distance of the robot to obstacles with the term $\phi_{\text{col}}(\boldsymbol{q})$. The cost weights c represent the importance of the corresponding subtasks. The parameter H in Equation (4) serves as a regularization parameter for smooth transitions.

Fusing many different task spaces together leads to the necessity that motion plans have to be adapted to the current situation. This leads to the two main characteristics of our approach:

1) Transformation of task space: The motion plan should be adapted to the current situation (e.g., the avoidance of an obstacle leads to a different motion shape).
2) Adaptation of phase space: The motion progress should depend on the state of the environment (e.g., if the goal moves far away, the duration should be increased).

Since some information is first available during motion execution, both adaptations are performed online at a high frequency rate.

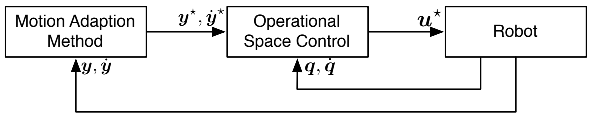


Fig. 3. Control diagram of the cascade controller.

In the following, we assume a motion execution framework like represented in Figure 3 with two execution loops. The inner loop is responsible for the operational space control in task space and will usually run with a high frequency. The outer loop runs with a lower frequency and provides new targets in task space for the inner loop. Furthermore, the outer loop is also responsible for adapting the plan to the current situation.

B. Online Task Space Transformation

The current plan in task space that the robot should follow is denoted with $\boldsymbol{\tau}^{\text{plan}}$ and parametrized over the phase variable $s \in [0,1]$. We transform this plan during execution to changes of the environment that differ from the static environment of the reference scenario. This plan is updated continuously during execution to two kinds of state changes: 1) The goal state changes. For example, the goal moves before or during execution of the movement or due to measurement variance the task space coordinates are updated. 2) The robot state is not on the plan. This situation can happen before execution if the robot has a different start state. It also naturally happens during execution as feedback regulators can never follow a plan exactly. Later we will consider an example wher an obstacle (that was unknown during planning) occurred on the path that needs to be avoided. The operational space controller accounts for the obstacle online and pushes the robot off the plan. Another source of perturbations is inaccuracy of the robot model, which we certainly have in the case of our PR2 demonstration, where the system does not accelerate as desired. We assume that we have a measurement of the current robot state $\boldsymbol{y}$ and the state of the goal $\boldsymbol{g}$ at each time step.

We propose to transform the current task space plan $\boldsymbol{\tau}^{\text{plan}}$ in such a way that two conditions are afterwards fulfilled:

1) The current state $\boldsymbol{y}$ of the robot is on the plan: $\boldsymbol{\tau}^{\text{plan}}(s) = \boldsymbol{y}$.
2) The current goal $\boldsymbol{g}$ is the last state in the plan: $\boldsymbol{\tau}^{\text{plan}}(1) = \boldsymbol{g}$.

For achieving these conditions, the plan $\boldsymbol{\tau}^{\text{plan}}$ is transformed for the remaining phase $\tilde{s} \in [s,1]$ with the update equation

$$\boldsymbol{\tau}^{\text{plan}}(\tilde{s}) = \boldsymbol{\tau}^{\text{plan}}(\tilde{s}) + \Delta\boldsymbol{g} + (\Delta\boldsymbol{g} - \Delta\boldsymbol{y}) \cdot \frac{\tilde{s}-1}{1-s}. \qquad (7)$$

Here, $\Delta\boldsymbol{y} = \boldsymbol{y} - \boldsymbol{\tau}^{\text{plan}}(s)$ is the distance that the robot is apart from the plan and $\Delta\boldsymbol{g} = \boldsymbol{g} - \boldsymbol{g}^{\text{prev}}$ is the change of the goal state.

The transformation properties are visualized in two-dimensional example motions in Figure 2. The upper graph in Figure 2(a) shows motions in a two dimensional task space. The red graph is the reference plan. The blue, green and black graphs are transformations of this reference plan to different target positions (represented as circles) and are computed with Equation (7). The upper graph in Figure 2(b) shows another demonstration of our transformation. Thereby, the goal moved during the motion execution (small dots show movement of goals). The transformation of the plan was performed continuously in each time step. The shape of the reference plan is maintained and transformed to the current situation.

C. Online Phase Adaptation

The phase variable is a substitute for time that measures progress in the plan. The value $s = 0$ means that the robot is at the beginning of the plan and $s = 1$ means that the target state has been reached. The phase variable is updated iteratively and always increases. In the following we propose a phase dynamics that allows the execution to progress with a speed that adapts to unforeseen events, such as an obstacle that slows down the operational space controller, or a goal transformation that elongates the distance to goal. We propose the phase dynamics

$$s = s + \Delta s^{\text{ref}} \cdot \underbrace{\frac{\|\boldsymbol{g}^{\text{ref}} - \boldsymbol{\tau}^{\text{ref}}(s)\|}{\|\boldsymbol{g} - \boldsymbol{y}\|}}_{\delta g} \cdot \underbrace{\frac{\|\boldsymbol{y} - \boldsymbol{y}^{\text{prev}}\|}{\|\boldsymbol{y}^\star - \boldsymbol{y}^{\text{prev}}\|}}_{\delta y}. \qquad (8)$$

In Equation (8), the variable Δs^{ref} is the reference step size, $\boldsymbol{y}$ is the current robot state, $\boldsymbol{y}^{\text{prev}}$ is the robot state of the previous time step, $\boldsymbol{y}^\star$ is the desired state of the current time step, $\boldsymbol{g}$ is the current goal, $\boldsymbol{g}^{\text{ref}} = \boldsymbol{\tau}^{\text{ref}}(1)$ is the reference goal state.

The progression of the phase variable in Equation (8) depends on two factors. The first factor δg is the ratio between the distance to the goal in the reference plan and the distance to the goal in the current situation. An intuition behind this ratio is that if the goal is moving away from us, the update of s should be smaller. The second factor δy in Equation (8) sets the distance of the previous actual step in relation with the distance of the previous planned step. If the planned step is larger than the actual step, then Equation (8) would lead to a smaller update of s.

As can be seen in Equation (8), if both factors are equal to 1 at each time step, we update the phase variable just with the reference step size Δs^{ref}, which would result in the exact same temporal trajectory as our reference plan $\hat{\boldsymbol{\tau}}^{\text{ref}}$. The case occurs when the goal never moves and the robot executes all control steps exactly as planned.

In Figure 2 the bottom graphs show the phase variable s plotted over time. For simplicity, we assumed in both examples that the robot is always able to execute the control

Algorithm 1 Adaptive Motion Execution (AMEX)

```
input:
    Update rate $\Delta t$
    Robot position $\boldsymbol{y}^{\text{prev}} = \boldsymbol{y}_0$; Goal position $\boldsymbol{g}^{\text{prev}} = \boldsymbol{g}_0$
    Reference trajectory $\hat{\boldsymbol{\tau}}^{\text{ref}}(t)$,   for $t \in [0, T^{\text{ref}}]$
initialization:
    Phase variable $s = 0$
    Reference step size $\Delta s^{\text{ref}} = \frac{\Delta t}{T^{\text{ref}}}$
    Remapped ref. trajectory $\boldsymbol{\tau}^{\text{ref}}(s) = \hat{\boldsymbol{\tau}}^{\text{ref}}(s \cdot T^{\text{ref}})$
    Planned trajectory $\boldsymbol{\tau}^{\text{plan}}(s) = \boldsymbol{\tau}^{\text{ref}}(s)$,   for $s \in [0, 1]$
while $s \leq 1$ run with frequency $\frac{1}{\Delta t}$
    1. Measure current state of system $\boldsymbol{y}$ and goal $\boldsymbol{g}$
    2. Update phase variable $s$                  (see Section III-C)
    3. Transform trajectory $\boldsymbol{\tau}^{\text{plan}}$           (see Section III-B)
    4. Compute next target state $\boldsymbol{y}^\star, \dot{\boldsymbol{y}}^\star$  (see Equation (9))
    5. Send next state to the task
       space controller                        (see Section III-A)
    6. Save last state $\boldsymbol{y}^{\text{prev}} = \boldsymbol{y}$ and $\boldsymbol{g}^{\text{prev}} = \boldsymbol{g}$
```

command exactly. In the motion visualized in Figure 2(a) the targets have an offset before the execution starts and do not move during execution. Here, the phase progress is linear with respect to time. The linear scaling constant is estimated with Equation (8) and is lower for targets that are further away. In Figure 2(b) the targets are initially all in the same position but move during the execution in different directions with different velocities. In such cases the phase progress is not linear anymore and depends on the movement direction and speed of the target. The black motion graph shows an example where the target first moves towards the robot and the phase variable progresses faster. Then the goal moves away from the robot and the phase variable progresses slower.

D. Adaptive Motion Execution Algorithm

With the task space execution method, the transformation of the task space and the phase adaptation we have all ingredients to execute adaptive motions on robots in dynamic scenarios. The Adaptive Motion Execution (AMEX) method combines all these single steps and is summarized in Algorithm 1.

The input of the algorithm is a reference trajectory $\hat{\boldsymbol{\tau}}^{\text{ref}}$ in some task space, which describes a motion of duration T^{ref}. This reference trajectory describes a task for a specific robot in a static environment scenario. Before starting the execution, the reference trajectory $\hat{\boldsymbol{\tau}}^{\text{ref}}$ is reparametrized from its parametrization over $t \in [0, T^{\text{ref}}]$ to a parametrization over the phase variable $s \in [0, 1]$. This reparametrization is done by simply defining $\boldsymbol{\tau}^{\text{ref}}(s) = \hat{\boldsymbol{\tau}}^{\text{ref}}(s \cdot T^{\text{ref}})$. The reference step size Δs^{ref} is set to $\frac{\Delta t}{T^{\text{ref}}}$, where Δt is the duration between replanning. The current plan is initialized with this reference plan $\boldsymbol{\tau}^{\text{plan}} = \boldsymbol{\tau}^{\text{ref}}$ and the robot and goal are put into a start state $\boldsymbol{y}_0$ and $\boldsymbol{g}_0$.

The while loop in Algorithm 1 contains the steps that are executed with a frequency of $1/\Delta t$. In step 1 of Algorithm 1 the current system state and goal state are measured and fed as input into the update of the phase variable in step 2, which is computed with Equation (8). Afterwards, the trajectory is transformed with Equation (7) such that the current robot and goal state are on the plan. In step 4 of Algorithm 1 the next target state is computed using the updated plan of Equation (7) according to

$$\boldsymbol{y}^\star = \boldsymbol{y} + \Delta s^{\text{ref}} \cdot \frac{\dot{\boldsymbol{\tau}}^{\text{plan}}(s)}{\|\dot{\boldsymbol{\tau}}^{\text{plan}}(s)\|} \cdot \|\dot{\boldsymbol{\tau}}^{\text{ref}}(s)\| \tag{9}$$

with the velocities $\dot{\boldsymbol{\tau}}^{\text{plan}}(s)$ of the current plan and $\dot{\boldsymbol{\tau}}^{\text{ref}}(s)$ of the reference plan. Thus, the robot moves into the direction of the current plan with the magnitude of the reference plan. Finally, this target state is passed on to the operational space control method and the current state is stored for the next iteration.

The steps 1–6 are repeated until the phase variable reaches the value 1, which denotes that the robot reached its goal.

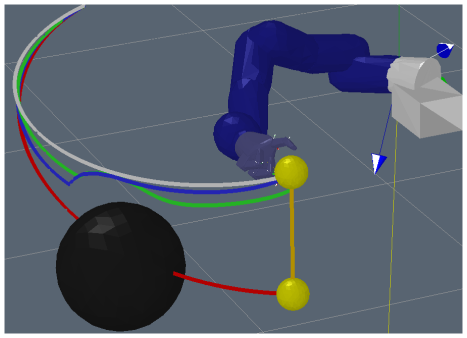


Fig. 4. This picture of the robot simulator shows a Task 3 scene of the simulation experiments (see Section IV-A). The black ball is an obstacle. The yellow balls represent the goal start and end state and the yellow line between is the goal movement during execution. The red line shows the reference plan $\boldsymbol{\tau}^{\text{ref}}$ that was created without any obstacle. The green (DMP), blue (AMEX), and white (CR) lines show the adapted motion $\boldsymbol{\tau}^{\text{plan}}$.

IV. Experiments

In the following section, we evaluate the performance of the AMEX method described in Algorithm 1 on simulated and real robot benchmark tasks. In all experiments, the reference plan is created with a trajectory optimizer that uses quadratic cost terms and a Gauss-Newton optimizer. We designed a cost function that is composed by the sum of squared joint accelerations, an obstacle avoidance term and the distance to the final robot state.

We executed this reference plan with operational space control (see Section III-A) and used as task spaces the position of the robot endeffector, the orientation of the robot endeffector, an obstacle avoidance term and a term to prefer smooth transitions in configuration space. We use the collision detection package SWIFT++ [1] to compute the distances to obstacles, which is integrated in the task vector in Equation (5).

A. Simulated Benchmark Tasks

In the following experiments, a simulated DLR light weight robot arm with convex shapes is used (see Figure 4). The robot has seven degrees of freedom and the goal of the

[1] `http://gamma.cs.unc.edu/SWIFT++`

		Success Rate	Final Time	Sum of Squared Acc.	Computational Time
Task 1	AMEX	1	4.17e+00 ±7.66e-01	6.52e+03 ±7.09e+03	1.02e-04 ±1.64e-05
	DMP	1	3.70e+00 ±8.58e-02	2.95e+04 ±4.33e+04	3.28e-05 ±1.96e-06
	CR	1	3.63e+00 ±5.01e-02	4.54e+03 ±6.46e+03	8.23e-03 ±1.24e-03
Task 2	AMEX	0.99	4.11e+00 ±1.15e+00	7.08e+03 ±7.27e+03	1.01e-04 ±1.48e-05
	DMP	0.99	5.77e+00 ±2.53e+00	1.73e+04 ±2.48e+04	3.39e-05 ±1.96e-06
	CR	0.99	3.65e+00 ±1.46e-01	5.65e+03 ±5.46e+03	1.22e-02 ±3.47e-03
Task 3	AMEX	0.91	5.32e+00 ±3.45e+00	3.97e+04 ±4.67e+04	9.24e-05 ±2.36e-05
	DMP	0.91	5.89e+00 ±3.56e+00	1.27e+05 ±3.79e+05	3.39e-05 ±3.53e-06
	CR	0.94	3.67e+00 ±5.40e-01	3.88e+03 ±5.83e+03	2.25e-02 ±8.51e-02
Task 4	AMEX	0.92	4.93e+00 ±3.22e+00	6.62e+04 ±2.30e+05	9.44e-05 ±3.09e-05
	DMP	0.93	5.86e+00 ±3.24e+00	2.17e+05 ±5.07e+05	3.97e-05 ±1.15e-04
	CR	0.92	3.72e+00 ±5.53e-01	4.23e+03 ±6.17e+03	2.27e-02 ±1.28e-01

Fig. 5. Evaluation results of the simulation task with mean and two times the standard deviation of the data.

task is that the robot reaches a desired target state with its endeffector. Thereby, changes in the environment happen and the robot has to adapt its motion.

We compare three methods for motion adaptation:
Adaptive Motion Execution (AMEX): This is the approach proposed in this paper based on task space transformation and phase adaptation (see Algorithm 1).
Dynamic Movement Primitives (DMP): This is an alternative phase space adaptation method mentioned in Section II-B. We implemented the standard formulation for static movements [10] and used Equation (1) for updating the phase variable.
Continuous Replanning (CR): This is a continuous replanning approach that uses a trajectory optimizer in each adaptation loop. It is the same trajectory optimizer with the same cost function that also created the reference plans for the DMP and AMEX method. Thereby, the previous plan is used as a starting point for the next iteration to speed up optimization.
All these methods are updated in an outer loop at 100 Hz and the operational space controller runs in an inner loop at 1 kHz (see Figure 3). In all tasks there is a fixed reference target position to which a reference plan is computed that is used as input in all motion adaptation methods. We created 4 benchmark tasks with different properties:
Task 1: The target was placed at a position that differs from the reference target position.
Task 2: Same as Task 1 plus the target moves during execution in a direction.
Task 3: Same as Task 2 plus there is an obstacle in the scene that the robot has to avoid.
Task 4: Same as Task 3 plus that the obstacle is also moving.
For each task 150 different environments were created. The position of the target, position of the obstacle, movement directions and movement speeds were thereby sampled from Gaussian distributions. An example scene is visualized in Figure 4. The red line represents the reference plan $\boldsymbol{\tau}^{\text{ref}}$. The yellow balls are the goal start and end position and the yellow line is the goal movement during the execution. The black ball is an obstacle that was put there after the planning phase. The AMEX motion is visualized as blue line, the DMP motion as green line and the CR motion as white line. The CR motion avoids the obstacle already very early, since it is included in the trajectory optimization cost function. The DMP and AMEX motions react later on the obstacle. It can be seen that the DMP motion first avoids the obstacle and then comes back to its original path. But the AMEX motion avoids the obstacle and transforms the plan such that it does not move down again, which comes closer to the behavior of the CR method.
The table in Figure 5 shows the results of the simulated experiments. We computed for each task and method different metrics that are given with the mean and two times the standard deviation over the 150 different environments. The success rate describes the ratio of in how many of the 150 environments the robot was able to reach the target state with a distance below 0.03 m. As it can be seen in the table, all methods reach a high success rate above 0.9, which shows that AMEX reaches a similar robustness to DMP and CR. The final time represents the duration of the actual robot motion. Our approach AMEX reaches a higher deviation in motion duration than CR, which comes along with the adaptive phase adaptation. So the motion takes longer if the goal moved away from the robot and vice versa. The sum of squared acceleration is computed from the resulting joint space accelerations. As expected, the continuous replanning approach reaches the best result. But our approach reaches a lower value than the DMP motions. The computational time is the time for one iteration of the computations that the adaptation method has to perform to compute the next target. As it can be seen, the CR approach takes longest, has a large deviation and increases with more complex tasks. Reducing the motion adaptation frequency would also lead to longer planning times for CR, since the changes in the environment will be bigger.

B. PR2 Reaching Motion

We implemented Algorithm 1 on a Willow Garage PR2 robot platform. We used the right arm of the PR2 for our experiments, which consist of seven rotational joints. The task was to reach a target position with the right hand of the PR2. The experimental setup is visualized in Figure 1. Each joint was controlled with a real-time torque controller at 1kHz. The motion adaptation loop communicated with this real-time controller at 20 Hz. We used the ROS package ar_track_alvar[2] for tracking the target state during execution. This package tracks the position of AR tags with the Kinect camera on the head of the robot. The duration of the motion was 10.6 s. The resulting motion is visualized in Figure 6.

[2]`http://wiki.ros.org/ar_track_alvar`

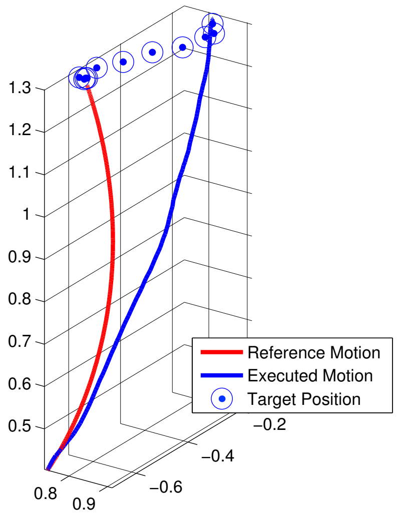


Fig. 6. The graphs shows motions of the PR2 experiments (see Section IV-B). The graphs show the motion of PR2s right hand in task space. The red line shows the reference motion $\boldsymbol{\tau}^{\text{ref}}$ created with a trajectory optimizer. The blue line shows the adapted executed motion $\boldsymbol{\tau}^{\text{plan}}$ of the robot to the new goal position. The blue dots represent the goal state movement during the motion execution.

The red line $\boldsymbol{\tau}^{\text{ref}}$ shows the reference motion of the PR2s right arm. The blue line represents the executed motion $\boldsymbol{\tau}^{\text{plan}}$, which was adapted during execution to the target change. The tracked targets are visualized as blue circles. As it can be seen, the transition from the reference motion to the adapted motion is smooth and works on real robots.

V. Conclusion

In this paper, we proposed an approach for continuously adapting pre-planned motions during their execution to changes in the environment. The motion plan is adapted in two manners: 1) The motion is transformed in task space, such that it is guaranteed that the robot and the goal are always located on the current plan; 2) The phase space is adapted in such a way that it relaxes the motion from a strict time dependency. We derived a phase update rule that includes the actual task progress and the change in goal location. We provided a concrete algorithmic implementation, which combines our approach with a low level task space controller. We demonstrated on simulated benchmark tasks that our method reaches a similar performance such as alternative motion adaptation methods. Additionally, we showed on a PR2 platform the real robot applicability of our approach, which is given through the low computational costs and the smooth transformation of the plan in task space. A limitation of the approach is to deal with situations where the goal moves out of the geometrical possible range of the robot or the obstacles are in such a constellation where no free path is possible anymore. In such cases, global replanning methods are required. In that respect, our approach bridges between such higher level planning methods and the fast control of motions. Future research might consider the prediction of future goal positions based on their past movement (e.g., with a Kalman filter) or the integration of additional environment feedback like contacts.

References

[1] O. von Stryk and R. Bulirsch, "Direct and Indirect Methods for Trajectory Optimization," *Annals of Operations Research*, vol. 37, no. 1, pp. 357–373, 1992.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A Survey of Robot Learning from Demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[3] G. Biggs and B. Macdonald, "A Survey of Robot Programming Systems," in *Proceedings of the Australian Conference on Robotics and Automation*, 2003.

[4] E. Yoshida, K. Yokoi, and P. Gergondet, "Online Replanning for Reactive Robot Motion: Practical Aspects," in *Proceedings of IROS*, 2010.

[5] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proceedings of ICRA*, 2006.

[6] O. Brock and O. Khatib, "Elastic Strips: A Framework for Motion Generation in Human Environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[7] C. E. García, D. M. Prett, and M. Morari, "Model Predictive Control: Theory and Practice — A Survey," *Automatica*, vol. 25, pp. 335–348, 1989.

[8] G. Klancar and I. Skrjanc, "Tracking-error Model-based Predictive Control for Mobile Robots in Real Time," *Robotics and Autonomous Systems*, vol. 55, no. 6, pp. 460–469, 2007.

[9] H. Diedam, D. Dimitrov, P.-B. Wieber, K. Mombaur, and M. Diehl, "Online Walking Gait Generation with Adaptive Foot Positioning through Linear Model Predictive Control," in *Proceedings of ICRA*, 2008.

[10] S. Schaal, "Dynamic Movement Primitives A Framework for Motor Control in Humans and Humanoid Robotics," *Adaptive Motion of Animals and Machines*, pp. 261–280, 2006.

[11] J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, 2008.

[12] J. Kober and J. Peters, "Imitation and Reinforcement Learning - Practical Algorithms for Motor Primitive Learning in Robotics," *IEEE Robotics and Automation Magazine*, no. 2, pp. 55–62, 2010.

[13] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives," in *NIPS*, 2013.

[14] S. Schaal, P. Mohajerian, and A. Ijspeert, "Dynamics Systems vs. Optimal Control — A Unifying View," *Progress in Brain Research*, vol. 165, pp. 425–445, 2007.

[15] J. K. Holm, D. Lee, and M. W. Spong, "Time-scaling Trajectories of Passive-dynamic Bipedal Robots," in *Proceedings of ICRA*, 2007, pp. 3603–3608.

[16] C. Chevallereau, "Time-scaling Control for an Underactuated Biped Robot," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 2, pp. 362–368, 2003.

[17] M. Al Borno and E. Fiume, "Feedback Control for Rotational Movements in Feature Space," *EUROGRAPHICS*, vol. 33, no. 2, 2014.

[18] T. Abbas and B. A. MacDonald, "Distance Indexed Trajectory Generation for a Helicopter Robot for Programming by Demonstration," in *Proceedings of IEEE International Conference on Advanced Intelligent Mechatronics*, 2009.

[19] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *Proceedings of ICML*, 2011.

[20] P. Englert, A. Paraschos, J. Peters, and M. P. Deisenroth, "Probabilistic Model-based Imitation Learning," *Adaptive Behavior Journal*, vol. 21, no. 5, pp. 388–403, 2013.

[21] M. P. Deisenroth and D. Fox, "Multiple-Target Reinforcement Learning with a Single Policy," *In ICML 2011 Workshop on Planning and Acting with Uncertain Models*, 2011.

[22] J. Z. Kolter, A. Coates, A. Y. Ng, Y. Gu, and C. DuHadway, "Space-indexed Dynamic Programming: Learning to Follow Trajectories," in *Proceedings of ICML*, 2008.

[23] S. Calinon, F. D'halluin, D. G. Caldwell, and A. G. Billard, "Handling of Multiple Constraints and Motion Alternatives in a Robot Programming by Demonstration Framework," in *Proceedings of Humanoids*, 2009.

[24] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1463–1467, 2008.

[25] O. Khatib, "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.