Reach tasks in QP robot control
task-points vs. SE(3) based formulations

van der Struijk, R.J.M.

Award date:
2019

Reach Tasks in QP Robot Control: Task-Points vs. SE(3) Based Formulations

Master's Thesis

CST 2019.052

Author:

R.J.M. van der Struijk BSc

Supervisor:

prof. dr. ir. H. Bruyninckx

Coach:

dr. ir. A. Saccon

Department of Mechanical Engineering
Control Systems Technology

June 18, 2019

Abstract

In the field of robotics, a fundamental task involves the positioning and orientation by parts of the robot. These parts are often the end-effector or a camera mounted on the robot. The types of tasks that need to reach for a certain desired position and/or orientation by a part of the robot are referred to as reach tasks in this thesis. It is observed that many of these reach tasks do not require all six degrees of freedom (DoF) to be specified. The state-of-the-art approaches on the formulation of reach tasks use a different representation of the current and desired pose depending on which DoF are specified by the reach task. Each representation also comes with its own error definition and PD controller strategy. This prompted the desire for a unified formulation using the same error definition and PD controller strategy for any reach task. The approach presented in this thesis uses a set of points that are rigidly attached to a part of the robot to represent the current pose. A second set of points is used to represent the desired location of the rigidly attached points. This approach is referred to as a task-point reach task. Different reach tasks can be specified by varying the amount of points and the relative location between these points. Each point uses the same error definition and PD controller. A single optimal control problem is created by embedding the cost function for each point in a task-based quadratic programming (QP) controller. Several basic reach tasks and combinations thereof are derived using this approach. A proof of concept simulation of the task-point reach task formulation is provided. The simulation shows the execution of a reach task that specifies all six DoF of the end-effector of a six joint serial manipulator.

Reach task formulations contain parameters, such as the PD gains of the controller that influence the resulting motion of the robot. A theoretical comparison with the state-of-the-art, an SE(3) based reach task, is made by studying the effects of these parameters on the resulting motion. This comparison shows that the parameters of the task-point reach task provide a more direct relation to the resulting motion. The task-point approach therefore offers a unified representation and a controller strategy that is intuitive in the specification of reach tasks, and hence eliminates the use of different representations and controller strategies.

Acknowledgements

I would like to take this opportunity to thank the people who have assisted me in the long journey that is called the graduation project. A long journey filled with opportunities to expand my knowledge and experience.

First of all, I would like to thank my direct supervisors Herman Bruyninckx and Alessandro Saccon for their support and critical feedback during the entirety of the project. I have learned to be more critical of existing literature and of my own work. As someone who very curious, I am also very appreciative of the one on one sessions where we would go more in-depth with a problem. Resulting in a greater understanding of the topic. I also appreciate that we would sometimes discuss interesting topics not always directly related to the project, but that piqued my interest.

Furthermore, I would like the thank the other students of the robotics lab. We organized several very fun activities, which set a positive mood in the lab and created many friendships. The breaks were never boring and provided the much needed mental break for our brains. I would like to especially thank Joep Linssen and Matthijs van der Burgh for helping me understand the software related problems I was having. I also thank Stefan Driessen and Casper Beumer for their support in sparring with me about QP related questions, and Martijn Bos for expanding my knowledge on Lie group related theory.

Finally, I would like to thank my family for their support, cooking meals for me when I came to visit and providing met with anti-stress cats and kittens to elevate my mood. I really appreciate this.

Rik van der Struijk
Eindhoven, June 2019

Nomenclature

A,B,C, ...	coordinate frames
$\boldsymbol{p}$	an arbitrary point
$\mathbf{o}_B$	origin of B
$[A]$	orientation frame associated to A
$B[A]$	frame with origin o_B and orientation $[A]$
${}^A\boldsymbol{p}$	coordinates of point $\boldsymbol{p}$ w.r.t. to A
${}^A\mathbf{o}_B$	coordinates of the origin $\mathbf{o}_B$ w.r.t. to A
${}^A\mathbf{R}_B$	rotation matrix relating $[B]$ to $[A]$
${}^A\mathbf{H}_B$	homogeneous transformation from B to A
${}^A\boldsymbol{X}_B$	velocity transformation from B to A
${}^C\mathbf{v}_{A,B}$	twist expressing the velocity of B w.r.t. to A written in C
${}^C\mathbf{v}^\wedge_{A,B}$	4×4 matrix representation of ${}^C\mathbf{v}_{A,B}$
${}^C\mathbf{v}_{A,B}\times$	6×6 matrix representation of the twist cross product
${}^C\boldsymbol{J}_{A,B}$	Jacobian relating the velocity of B w.r.t. A expressed in C

Contents

Chapter 1

Introduction

1.1 Reach Tasks in Robot Control

In the field of robotics, a fundamental task involves the positioning and orientation by parts of the robot. These parts are often the end-effector or a camera mounted on the robot and are used in manipulation and sensor tasks, see for example Figure 1.1. Both examples show that the end-effector or camera need to be positioned and oriented in one or more stages of the specific task. The types of tasks that need to reach for a certain desired position and/or orientation by a part of the robot are referred to as reach tasks in this thesis. Another observation is that not all reach tasks require all six degrees of freedom (DoF) to be specified. Take for example Amigo picking up the can in Figure 1.1a. As long as the arm does not collide with itself or the can, the rotation about the axis perpendicular to the table does not need to be specified. The gripper will be able to pick up the can for any rotation about this axis, and hence leaves this DoF free. These types of reach tasks are referred to as redundant reach tasks.

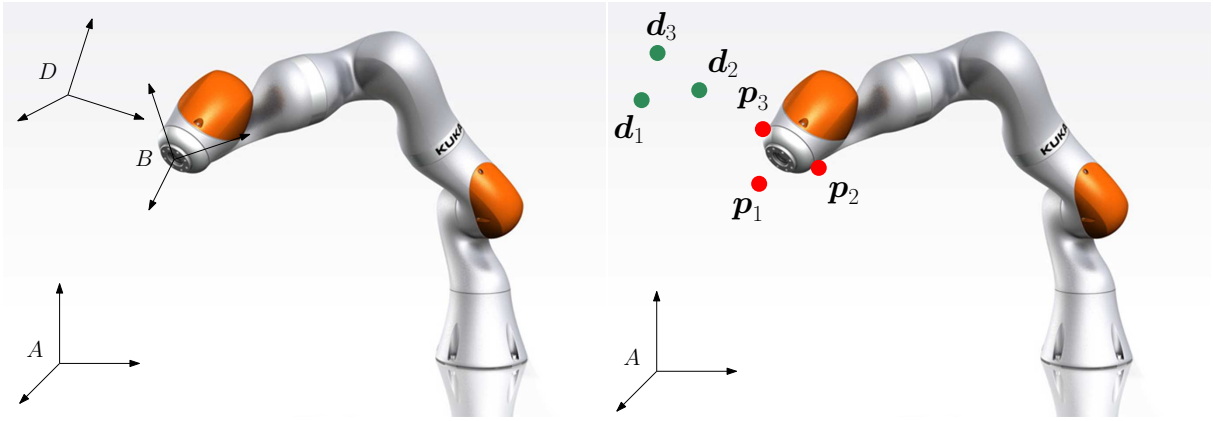
(a) *Domestic robot Amigo from Tech United (TU/e) [1]*

(b) *Camera robot Iris from Bot & Dolly [2]*

Figure 1.1: *Two examples of robotic systems executing (a) a manipulation + sensor task and (b) a sensor task.*

To specify a (redundant) reach task the current and desired pose of a part of the robot requires a mathematical representation. Consider for example the robot arms in Figure 1.2. The figure shows a similar reach task for both robot arms, however, the representation of the current pose

and desired pose differ. A coordinate frame defined by the combination of an origin $\mathbf{o}_B$ and an orientation frame $[B]$ such that $B = (\mathbf{o}_B, [B])$. In Figure 1.2a, the coordinate frame B is rigidly attached to the end-effector and is used to describe the current pose. Frame D describes the desired pose of the end-effector. Both frames can be expressed relative to the inertial frame A using a homogeneous transformation matrix ${}^{A}\mathbf{H}_{B} \in \mathrm{SE}(3)$ of a frame B with respect to frame A, where SE(3) is the special Euclidean group of dimension three. Since the homogeneous transformation matrix is an element of SE(3), this reach task is referred to as an SE(3) reach task. In Figure 1.2b the three points $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_3$, which are rigidly attached to the end-effector, are used to describe the current pose. Each of these points has a corresponding desired point $\boldsymbol{d}_i$. Both sets of points can be expressed in the inertial frame A, as ${}^{A}\boldsymbol{p}_i$ and ${}^{A}\boldsymbol{d}_i$. This type of reach task is referred to as a task-point reach task. The task-point reach task is the approach investigated in this thesis to specify reach tasks as an alternative to the 'standard' approaches in literature, which are discussed in the next section.



(a) (b)

Figure 1.2: *Visualization of (a) an SE(3) reach task and (b) a task-point reach task.*

In most robotic applications there is no direct control over a part of the robot in all six DoF, instead multiple joints connected to each other create an n DoF kinematic chain, see for example the robot arms in Figure 1.2. The core problem of a reach task is therefore to generate joint commands such that, from its initial configuration, a part of the robot eventually reaches its desired position and/or orientation in task-space (Cartesian space). An approach to this problem is to consider the separate components of the problem, the generation task-space trajectory, the conversion to joint space trajectories, the robot and control of the joints as individual problems. Consider for example the end-effector of a robot arm. First, a reference trajectory for the end-effector in task-space is generated that moves the end-effector from its initial pose to its desired pose. Then, inverse kinematics (IK) is used to map the desired task-space trajectory to joint space trajectories. Note that this map is not always unique or feasible and thus it is possible to have multiple or no solutions. Several IK methods can be found in [3]. The map of a task-space trajectory to joint space trajectories is never unique for redundant manipulators, where the task requires less DoF than the manipulator offers. Thereafter, a low level joint controller is used to realize the joint trajectories, which is often a regular PID type of controller.

The approach just described highlights the essential components in executing a reach task. However, it considers each component as a separate problem, which might lead to infeasible

solutions. For example, the kinematics of the robot is only considered after generating the task-space trajectory, which might lead to joint space trajectories that violate bounds set on the joints. Additionally, the task-space trajectory is generated off-line as a path planning problem. However, the robots executing these reach tasks are increasingly put into dynamic environments where also humans are present. For example, domestic robots are supposed to support sick, disabled or elderly people by serving drinks or breakfast, storing the groceries on a shelf or taking out the trash [4]. Robots in these kinds of environments therefore have to consider the dynamic environment as part of the problem and hence require the on-line adaptation of the task-space trajectory.

Another approach is to consider all components together as a single problem by viewing the reach task as an optimal control problem. A reach task is specified in task-space, the term task-based control or output-based control is therefore used to refer to a control strategy that uses feedback on the current state of the robot in task-space. Task-based controllers such as the reactive QP controllers in [5, 6] generate both the task-space and joint space trajectories simultaneously and on-line. The output of the QP controller are joint space trajectories and therefore also require a low level joint controller to realize the joint trajectories. Due to the reactive nature of the QP controller it is possible to consider a dynamic environment as part of the reach task problem and not separately. For this reason, reactive or predictive control solutions are preferred over off-line path planning and then tracking solutions. Additional challenges are to be considered when performing a reach task on a real physical robot. Relevant challenges are configuration singularities, self-collisions, the dynamics of the robot and physical limits such as joint bounds and actuator saturation. Reactive task-based QP controllers are also able to take into account these additional challenges as multiple tasks and/or constraints to the QP problem, as shown in [5, 6].

Besides additional tasks as a result of dealing with the challenges of using a physical robot, robots are also often tasked with multiple (reach) tasks simultaneously. Consider for example again Figure 1.1a, Amigo executes two tasks simultaneously, one is the end-effector reaching for the can and the other is the gaze direction of the camera mounted on top of the robot. Both tasks are required to successfully pick up or place the can. In this case the tasks are not conflicting. However, when for example a collision avoidance task for the robot arm is added, it may be conflicting with the manipulation task.

The discussion in this section shows that there is a need for a formulation in which multiple, possibly conflicting, (redundant) reach tasks can be formulated. The approach of this thesis is therefore to assess existing control formulations that treat the (redundant) reach task as a task-based control problem. This problem is explored in more detail in the next section.

1.2 Existing Methods for Reach Task Formulations and Multi-Task Robot Control

The objective is to create a reactive control formulation in which multiple, possibly conflicting, (redundant) reach tasks can be incorporated and the manipulator redundancy is resolved. This problem is split into two parts: the formulation of reach tasks and multi-task robot control. The formulation of a reach task, consists of a representation for the current and desired pose, an error definition and a control strategy that drives the error to zero.

1.2.1 Pose Representations

The term pose is used to represent the position and orientation of a part of the robot. The position is uniquely described by a vector in $\mathbb{R}^3$, orientation however, is not described by a vector space, but by a Lie group. Specifically, the special orthogonal group of dimension three SO(3), which is the set of $\mathbb{R}^{3\times3}$ orthogonal matrices with determinant equal to one endowed with matrix multiplication. The rotation matrix ${}^{A}\mathbf{R}_{B}$ is an element SO(3) and describes the rotation from orientation frame $[B]$ to $[A]$. The rotation matrix can therefore be used to describe the orientation of a frame. As discussed in the previous section, the pose is described using ${}^{A}\mathbf{H}_{B} \in \mathrm{SE}(3)$, where SE(3) is also a Lie group. The rotation matrix is not the only way to describe the orientation, parameterizations of SO(3) also exist [7, 8]. For example, Euler angles, quaternions and axis-angle pair, which are briefly discussed in the next paragraph.

The rotation matrix description uses nine parameters while only three DoF are involved. Minimal representations use only three parameters (e.g. Euler angles), however, these representations suffer from singularities in one way or another. For example, Euler angles are singular in the transformation from the time derivative of the angles $\dot{\boldsymbol{\theta}}$ to the angular velocity $\boldsymbol{\omega}$ of a frame. This particular singularity is also referred to as a gimbal lock in [7] and corresponds to a singularity in the representation Jacobian. Due to this singularity, the time derivatives of the Euler angles are not globally defined. The Euler angle representation also suffers from non-uniqueness, since multiple angles map to the same rotation (e.g. 0 rad and 2π rad represent the same rotation). Continuous time control using Euler angles should only be used if the requested angular motions are small and not in the neighborhood of a representational singularity. Since all minimal representations suffer from singularities, they are not suitable for continuous time control or are limited to local motions away from singularities. Other non-minimal parameterizations (e.g. quaternions and axis-angle pairs) use an additional k parameters, related by k constraints in order to keep the original three DoF of the system. Both quaternion and axis-angle pair representations are globally defined, however, their non-uniqueness may lead to undesired effects such as unwinding [7, 9].

Reduced Pose Representations. As discussed in the first section, redundant reach tasks do not specify all six DoF of a pose, instead one or more DoF is/are left free. The representations of the current and desired pose for such reach tasks are referred to as reduced pose. When a translational DoF is left unspecified, the vector representing a reduced position becomes an element of $\mathbb{R}^2$ or $\mathbb{R}$. In the case of orientation, the reduced orientation representation of [7, 10] can be used. The reduced orientation representation uses a unit vector representing the pointing direction in $\mathbb{S}^2$ (the unit sphere in $\mathbb{R}^3$), as shown in Figure 1.3. This figure shows the unit vector $\boldsymbol{v} \in \mathbb{S}^2$, which can be represented with respect to an inertial frame A in $\mathbb{R}^3$.

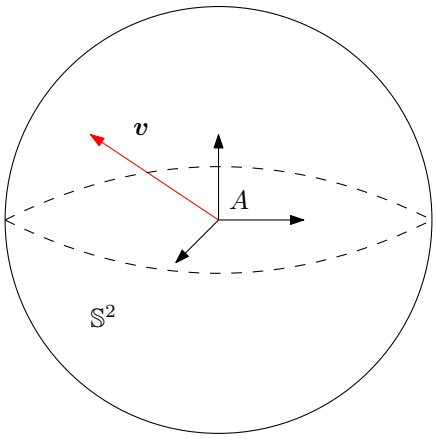


Figure 1.3: *Illustration of a reduced orientation representation using the unit vector* $\boldsymbol{v} \in \mathbb{S}^2$*, which can be expressed in the inertial frame* A *in* $\mathbb{R}^3$*.*

This representation leaves the rotation about the pointing direction free and hence can be used in pointing or grabbing applications (e.g. Figure 1.1a). Other parameterizations, such as Euler angles, quaternions and axis-angle can also be used a reduced attitude representations, however, also still suffer from the same non-global and non-uniqueness issues discussed above.

From this discussion it becomes clear that the rotation matrix and reduced attitude vector of [7, 10] are orientation representations that are global and unique and hence fit for continuous time control. Therefore, in the remainder of this literature review, only representations in SO(3) and in $\mathbb{S}^2$ are considered.

1.2.2 Pose Error Definition & Control

The pose is described by the homogeneous transformation matrix, which is an element of SE(3), and control on this pose has two approaches as studied in [11]. The first approach is PD control on $\mathrm{SO}(3) \times \mathbb{R}^3$, which comes down to separately applying PD control on $\mathbb{R}^3$ and PD control on SO(3). This type of control is referred to as double-geodesic control in [11]. The second approach is PD control on SE(3) directly. Starting with the first approach, [11] describes a double-geodesic PD feedback control function

$$u_{\mathbb{R}^3} : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3 \tag{1.1}$$

and

$$u_{\mathrm{SO}(3)} : \mathrm{SO}(3) \times \mathbb{R}^3 \to \mathbb{R}^3 \tag{1.2}$$

that almost globally stabilizes a desired pose equilibrium. The position error is defined as the standard vector error between the current and desired position of a part of the robot. The velocity error is the time derivative of the position error. The orientation error is defined as the product of the desired rotation matrix transposed with the rotation matrix representing the current orientation of a part of the robot. This error is an element of SO(3), however, (1.2) shows that the for a control application the output should be in $\mathbb{R}^3$. For this reason the orientation error is transformed to its Lie algebra $\mathfrak{so}(3)$ by means of a logarithmic mapping. The result is a skew-symmetric 3×3 matrix, which has only three unique values and hence can be written as a vector in $\mathbb{R}^3$. The angular velocity error is the vector error between the current and desired

angular velocity of a part of the robot.

The second approach in [11] describes a PD feedback control function

$$u_{\mathrm{SE}(3)} : \mathrm{SE}(3) \times \mathbb{R}^6 \to \mathbb{R}^6 \tag{1.3}$$

that almost globally stabilizes a desired pose equilibrium. The pose error is the product of the desired homogeneous transformation matrix transposed with the homogeneous transformation matrix representing the current pose of a part of the robot. Again a logarithmic map is used to transform this error to its Lie algebra $\mathfrak{se}(3)$ and can be written as vector in $\mathbb{R}^6$. The velocity error is defined as the vector error between the current and desired twist (velocity vector in $\mathbb{R}^6$).

Reduced Pose. The reduced position is described as a vector in $\mathbb{R}^2$ or $\mathbb{R}$ therefore the PD feedback control functions become

$$u_{\mathbb{R}^2} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2 \tag{1.4}$$

and

$$u_{\mathbb{R}} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \tag{1.5}$$

respectively. The reduced orientation is described by a vector in $\mathbb{S}^2$. Feedback control of the reduced orientation has been studied in [10, 7] as PD control on $\mathbb{S}^2$ and describes a feedback control function

$$u_{\mathbb{S}^2} : \mathbb{S}^2 \times \mathbb{R}^3 \to \mathbb{R}^3 \tag{1.6}$$

that stabilizes a desired reduced orientation equilibrium. The reduced orientation error in [7] is the cross product of the desired reduced orientation with the current reduced orientation of a part of the robot.

1.2.3 Multi-Task Robot Control

The example in Figure 1.1a shows that robots often have to deal with multiple, possibly conflicting, (redundant) reach tasks using redundant manipulators. The goal is therefore to assess control formulations that can cope with these challenges. In [12, Chapter 5] different state-of-the-art approaches to multi-task robot control have been compared. The approaches can be captured into two main categories: null-space projection methods and quadratic programming (QP) methods. Both methods view a task as a constrained optimal control problem, specifically a QP problem.

Redundancy. An n DoF manipulator is considered redundant when the m DoF task it needs to execute specifies fewer DoF than the manipulator offers (i.e. $n > m$). Specifically, it is redundant with respect to the task and hence is also referred to as task redundancy, where $n - m$ is the task redundancy degree.

QP Control. Consider for example the two, possibly, conflicting point reach tasks, as shown in Figure 1.4. The objective of reach task one is to match the body-fixed point $\boldsymbol{p}_1$ with the corresponding desired point $\boldsymbol{d}_1$. The objective of reach task two is to match body-fixed point $\boldsymbol{p}_2$ with the corresponding desired point $\boldsymbol{d}_2$. The two reach tasks can be conflicting in two cases. Firstly, the manipulator physically can't reach both tasks simultaneously, which depends on the

location of both $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$. Secondly, the manipulator is not redundant with respect to both tasks (i.e. $n < (m_1 + m_2)$), meaning that the manipulator is not able to comply with both reach tasks even if they are physically obtainable. Unless, for example, the desired point of reach task two coincides with the solution of reach task one. The task redundancy degree in this case is equal to one, since both point reach tasks specify three DoF, and the manipulator has seven DoF.

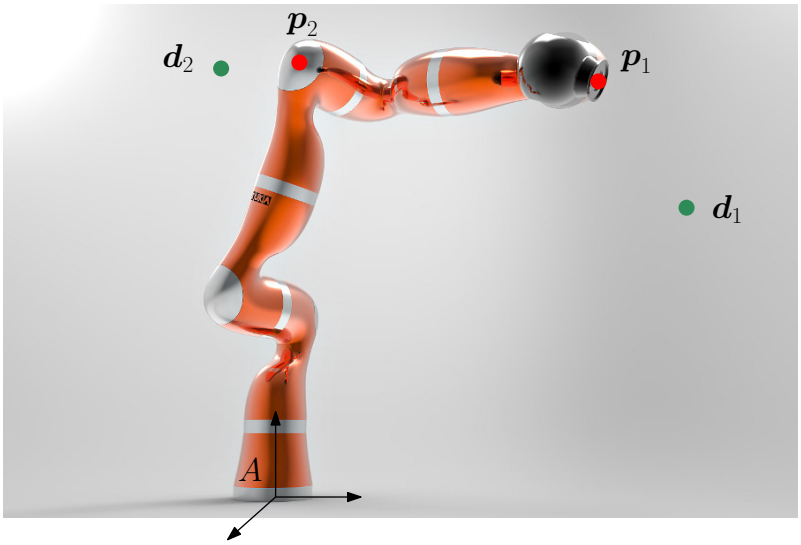


Figure 1.4: *Visualization of two point reach tasks to be performed by a seven DoF manipulator.*

The derivation of a general QP controller is discussed in Chapter 2. For now, consider a point reach task to be formulated as a QP controller on velocity level (similar to [13]) as

$$\begin{aligned} \min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \quad & \frac{1}{2} || {}^A\dot{\boldsymbol{p}} - \boldsymbol{u} ||^2 \\ \text{s.t.} \quad & {}^A\dot{\boldsymbol{p}} = {}^A\boldsymbol{J}\dot{\boldsymbol{q}} \\ & \boldsymbol{u} = -K_p({}^A\boldsymbol{p} - {}^A\boldsymbol{d}) + {}^A\dot{\boldsymbol{d}}, \end{aligned} \tag{1.7}$$

where the dot notation represents the time derivative d/dt, n the number of joints, $\boldsymbol{q}$ the joint angles, ${}^A\boldsymbol{J} := \partial {}^A\boldsymbol{p}_i(\boldsymbol{q})/\partial \boldsymbol{q} \in \mathbb{R}^{3\times 3}$ the Jacobian expressed in frame A, $\boldsymbol{u}$ the output of the feedback and feedforward controller and $K_p \in \mathbb{R}^{3\times 3}$ the gain matrix. Or written compactly as

$$\min_{\dot{\boldsymbol{q}} \in \mathbb{R}^n} \quad \frac{1}{2} || {}^A\boldsymbol{J}\dot{\boldsymbol{q}} + K_p({}^A\boldsymbol{p} - {}^A\boldsymbol{d}) - {}^A\dot{\boldsymbol{d}} ||^2. \tag{1.8}$$

The result is a task-based controller formulated as a QP problem, where the task error, under the influence of constraints, is to be minimized in order to execute this task. The QP problem resolves the task redundancy by minimizing the associated task error as the cost function and hence "chooses" a solution. The output of this particular QP controller (i.e. the solution to the QP problem) are the joint velocities $\dot{\boldsymbol{q}}^* \in \mathbb{R}^n$. Since the input is in task-space and the output in joint space, the QP controller effectively solves the inverse kinematics (IK) problem. Integration of $\dot{\boldsymbol{q}}^*$ is then performed to retrieve the updated state of the robot arm $\boldsymbol{q}^*$ and is used to update the kinematic model for the next time-step. Besides control in task-space, the joint references $\boldsymbol{q}^*$ and $\dot{\boldsymbol{q}}^*$ should also be realized in e.g. the presence of disturbances or model errors. Therefore, joint space control is also part of the problem and often regular PID type controllers are

used for this application. Some tasks, like joint bounds, are naturally described using inequality constraints. The QP formulation of (1.7) naturally allows for inequality constraints, since a numerical solver is employed to solve the optimal control problem.

The example in Figure 1.4 shows two reach tasks that are possibly conflicting. The solutions to this possible conflict can be divided into three main categories. The first category considers task one as the main task and only allow motions towards the second task that do not disturb the main task. The second category considers both tasks simultaneously with a relative importance. The third category determines which task should be active based on a certain condition, for example, first complete reach task one and then two. The first two categories are a form of hierarchy, hard and soft priority respectively. The third category is a form of task switching based on either simple conditions or possibly a higher level decision making process.

Hard Priority. In a QP context, hard priority is realized by successively solving QP problems while ensuring that the lower priority task is solved in the remaining solutions space of the higher priority task as is done in [14]. The result is

$$\begin{aligned} \min_{\dot{\boldsymbol{q}}_2 \in \mathcal{S}} \quad & \frac{1}{2}||{}^{A}\boldsymbol{J}_2\dot{\boldsymbol{q}}_2 + K_{p_2}({}^{A}\boldsymbol{p}_2 - {}^{A}\boldsymbol{d}_2) - {}^{A}\dot{\boldsymbol{d}}_2||^2 \\ \text{s.t.} \quad & \mathcal{S} = \left\{ \arg\min_{\boldsymbol{q}_1 \in \mathbb{R}^n} \ \frac{1}{2}||{}^{A}\boldsymbol{J}_1\dot{\boldsymbol{q}}_1 + K_{p_1}({}^{A}\boldsymbol{p}_1 - {}^{A}\boldsymbol{d}_1) - {}^{A}\dot{\boldsymbol{d}}_1||^2 \right\}, \end{aligned} \tag{1.9}$$

where $\mathcal{S}$ is the remaining solutions space after solving for reach task one. In [13] the remaining solutions space $\mathcal{S}$ is directly implemented as the equality constraint ${}^{A}\boldsymbol{J}_1\boldsymbol{q}_2 = {}^{A}\boldsymbol{J}_1\boldsymbol{q}_1^*$. This constraint can be interpreted as stating that the second reach task should still comply with the solution of the first reach task ${}^{A}\dot{\boldsymbol{p}}_1 = {}^{A}\boldsymbol{J}_1\boldsymbol{q}_1^*$.

The null-space projection method is based on the same principle as the successive QP problems, the lower priority task should not interfere with the higher priority task. In the context of multi-task robot control a null-space projection can be interpreted as a lower priority task that is applied to the null-space of a higher priority task. This can be done repeatedly to create a hierarchy. The strictness of this hierarchy is measure for how much and how a lower priority task influences a higher priority task. A comparison of null-space projectors and their effect on the strictness of the hierarchy is made in [15]. The main result is that an augmented mass matrix weighted null-space projection is required for a strict hierarchy, which for example, is used on the HRP-2 humanoid robot in [16]. Note that a hard priority hierarchy of tasks only makes sense when the higher priority task leaves a non-empty null-space for additional tasks. Consider for example the end-effector of an n DoF manipulator, if the end-effector is required to execute a task that requires n DoF, then there are no DoF "left" (i.e. an empty null-space) for additional tasks.

In the case of inequality constraints, null-space projection methods are limited, since they need special treatment. In [17] a hierarchical active-set algorithm is developed where inequality constraints are added as equality constraints when they are considered active by the algorithm.

Soft Priority. In a QP context, soft priority is realized by adding the cost of each task using weights to signify the relative importance between tasks, as is done in [6]. On velocity level this

becomes

$$\min_{\dot{\boldsymbol{q}}\in\mathbb{R}^n} \quad \frac{1}{2}\sum_{i=1}^{2} w_i||{}^A\boldsymbol{J}_i\dot{\boldsymbol{q}}_i + K_{p_i}({}^A\boldsymbol{p}_i - {}^A\boldsymbol{d}_i) - {}^A\dot{\boldsymbol{d}}_i||^2, \tag{1.10}$$

where w_i is the relative weight between the two tasks. In a general task context the combination of a hard and soft QP hierarchy is also possible. However, one should determine to what extent that is useful for the task.

Task Switching. The last strategy to cope with conflicting tasks is to switch tasks based on simple conditions or a possibly a higher level decision making process. Using the situation of Figure 1.4, reach task one could, for example, be execute first using a QP controller with the condition that the Euclidean norm of the task error is smaller than some constant epsilon (i.e. $||{}^A\boldsymbol{p}_1 - {}^A\boldsymbol{d}_1||_2 < \epsilon$). And then switching to the second reach task that also uses a QP controller. The same task switching strategy can applied to the null-space projection case. Take for example a situation where two tasks are implemented using a null-space projection and a third task e.g. obstacle avoidance is triggered by a certain distance condition and needs to take full priority. When a set of tasks gets more elaborate/complex and the redundancy or null-space might be empty, one might want to consider a switching strategy in order to preserve predictability of the resulting motion of the robot arm for a set of tasks.

In conclusion, two main multi-task approaches, QP control and null-space projection, have been compared and both approaches are able to provide a formulation for multiple, possibly conflicting, (redundant) reach tasks used on redundant manipulators. This has been shown in the form of hard and soft priority hierarchy and a task switching approach. The QP approach offers the most flexible formulation due to the fact that inequality constraints can be added to the problem without additional care and because it can also be used as a soft priority hierarchy.

1.3 Research Objectives

The literature review of the previous section shows that the most elaborate multi-task formulation is constructed by viewing a task as a QP problem. The discussion on the formulation of a reach task has led to multiple different representations depending on the amount of DoF used for a reach task. Each representation requires a different type of error definition and controller strategy. It is therefore interesting to explore a way of creating a reach task formulation that uses the same representation irrespective of the amount DoF specified by the reach task. The error definition and control strategy should also be independent of the amount DoF specified by the reach task (i.e. a unified reach task formulation). It is then interesting to compare this formulation with a state-of-the-art formulation. The SE(3) representation with PD control on SE(3) is chosen as the state-of-the-art, because it considers the structure of the SE(3) Lie group as part of the problem and can cover all reach tasks with an error definition that is inertial frame independent. However, this approach does disregards the DoF that are left "free" by redundant reach tasks, because the pose error cannot be split into smaller parts. This thesis only considers the kinematic aspects of the QP controllers, because the focus is on reach task formulations and the convergence of the QP controller towards its the goal. This leads to the main research objective of this thesis.

'Create a unified reach task formulation, compare this with a state-of-the-art reach task formulation in a QP control context and provide a proof of concept simulation of the unified reach task formulation.'

To achieve this objective, five sub-objectives are defined:

1. **Understand the state-of-the-art reach task formulation and structure the obtained knowledge**
   Investigate PD control on SE(3), how to embed it in a QP controller and how to formulate a reach task. The output of a QP controller on acceleration level are joint acceleration trajectories and usually requires a joint space controller. However, a purely kinematic simulation considers each joint as a decoupled double integrator system. Therefore, the QP solution is directly applied to the system and as a consequence, the system states are updated by integrating twice.

2. **Create a unified reach task formulation**
   Create a unified reach task formulation in the form of a QP controller, asses its ability to specify (redundant) reach tasks and assess the effects of its parameters on the resulting motion.

3. **Compare the state-of-the-art with the unified approach**
   Make a theoretical comparison of both methods on their ability to specify (redundant) reach tasks and their ability to tune the resulting motion.

4. **Software implementation of the unified reach task formulation**
   Implement the unified reach task formulation in either an existing software framework or create one.

5. **Provide a proof of concept simulation of the unified reach task formulation**
   The proof of concept should show the successful execution of a reach task using the unified reach task formulation and analyze the resulting motion.

1.4 Approach

In an effort to create a unified reach task formulation inspiration is found in [18]. Herein, a task-function approach is used as a general formulation for sensor-based control of robot manipulators. In one of their examples a sensor task is shown using three sensors in a plane surface following task. From this example, the idea was born to use a set of points $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N$ that are rigidly attached to a part of the robot to be matched by a second set of corresponding desired points $\boldsymbol{d}_1, \cdots, \boldsymbol{d}_N$ in $\mathbb{R}^3$. An example of three points is shown in Figure 1.2b. The set of points indirectly represents the position and/or orientation of a part of the robot and can therefore be considered a reach task. Since this idea is inspired by the task-function approach and makes use of points, this type of reach task is referred to as a task-point reach task. The task-point reach task presents a unified representation for multiple different (redundant) reach tasks by varying the amount and placements of both sets of points. The specification of task-point based reach tasks and its formulation as a QP controller are discussed in Chapter 3.

1.5 Report Outline

The report is structured as follows. Chapter 2 presents essential background material and introduces the multi-body dynamics notation used throughout this report. Chapter 3 formulates the SE(3) and task-point based reach tasks as a QP controller. At the end of this chapter a theoretical comparison between both methods is made. Chapter 4 provides a proof of concept simulation of a task-point reach task. Finally, in Chapter 5 conclusions on the research are drawn and areas for future research are presented.

Chapter 2

Preliminaries

This chapter provides a review of relevant preliminary material for this thesis. Starting with the multibody dynamics notation used throughout this thesis. Optimization-based control and the controller design for different representations of pose are treated next. Finally, the software architecture, and its components, that enables numerical simulations of reach tasks is introduced.

2.1 Multibody Dynamics Notation

This section proves a review of the multibody dynamics notation introduced in [19]. First, points and coordinate frames are discussed. Second, velocity vectors in different frames and coordinate transformations are presented. Third, acceleration vectors are derived and finally the forward kinematics are reviewed.

2.1.1 Points and Coordinate Frames

A coordinate frame A is defined as the combination of a point $\mathbf{o}_A$ (called origin) and an orientation frame $[A]$ in 3D space, such that $A := (\mathbf{o}_A,[A])$. The coordinates of a point $\boldsymbol{p}$, with respect to frame A, are collected in the coordinate vector ${}^A\boldsymbol{p}$. This vector represents the 3D geometric vector $\vec{r}_{\mathbf{o}_A,\boldsymbol{p}}$ connecting the origin of frame A with the point $\boldsymbol{p}$, pointing towards $\boldsymbol{p}$, expressed in the orientation frame $[A]$. Mathematically, this is written as

$$ {}^A\boldsymbol{p} := \begin{bmatrix} \vec{r}_{\mathbf{o}_A,\boldsymbol{p}} \cdot \vec{x}_A \\ \vec{r}_{\mathbf{o}_A,\boldsymbol{p}} \cdot \vec{y}_A \\ \vec{r}_{\mathbf{o}_A,\boldsymbol{p}} \cdot \vec{z}_A \end{bmatrix} \in \mathbb{R}^3, \tag{2.1} $$

where $\cdot$ denotes the scalar product between two vectors and $\vec{x}_A$, $\vec{y}_A$, $\vec{z}_A$, are the unit vectors defining the orientation frame $[A]$. Figure 2.1 shows an illustration of two frames A, B, a point $\boldsymbol{p}$ and the relations between them.

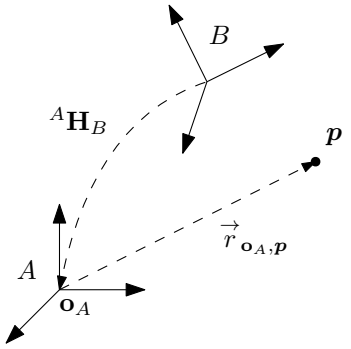


Figure 2.1: *Illustration of two frames A and B, and a point $\boldsymbol{p}$.*

The rotation matrix ${}^{A}\mathbf{R}_{B} \in \mathrm{SO}(3)$ is used to denote the coordinate transformation from frame B to frame A and is only dependent on the orientation frames $[A]$ and $[B]$, irrespectively of the position of the origins $\mathbf{o}_A$ and $\mathbf{o}_B$. The 4×4 homogeneous transformation matrix

$$ {}^{A}\mathbf{H}_{B} := \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{o}_{B} \\ 0_{1\times 3} & 1 \end{bmatrix} \in \mathrm{SE}(3) \tag{2.2} $$

is used to express the position and orientation of frame B with respect to frame A. The homogeneous transformation matrix can also be used to map to the coordinate vector ${}^{B}\boldsymbol{p}$ to ${}^{A}\boldsymbol{p}$. To this end, a homogeneous representation of ${}^{A}\boldsymbol{p}$ is defined as ${}^{A}\bar{\boldsymbol{p}} := ({}^{A}\boldsymbol{p}; 1)$ (the symbol ; denotes row concatenation). The map is then given by

$$ {}^{A}\bar{\boldsymbol{p}} = {}^{A}\mathbf{H}_{B}\, {}^{B}\bar{\boldsymbol{p}}. \tag{2.3} $$

2.1.2 Velocity Vectors (Twists)

The the hat operator $\wedge$ is used to map a vector $\mathbf{w} \in \mathbb{R}^3$ to a skew-symmetric matrix in $\mathfrak{so}(3)$, such that

$$ \mathbf{w}^{\wedge} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{\wedge} := \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \in \mathfrak{so}(3). \tag{2.4} $$

In the case of a vector $\mathbf{v} = (\boldsymbol{v}; \boldsymbol{\omega}) \in \mathbb{R}^6$, with $\boldsymbol{v}$ and $\boldsymbol{\omega} \in \mathbb{R}^3$, the hat operator $\wedge$ maps the vector $\mathbf{v}$ to a matrix in $\mathfrak{se}(3)$, such that

$$ \mathbf{v}^{\wedge} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}^{\wedge} := \begin{bmatrix} \boldsymbol{\omega}^{\wedge} & \boldsymbol{v} \\ 0_{1\times 3} & 0 \end{bmatrix} \in \mathfrak{se}(3). \tag{2.5} $$

The vee operator $\vee$ is used to indicate the inverse operation. The velocity of frame B with respect to frame A can be expressed in several different frames. The left trivialized velocity of frame B with respect to A is given by the twist

$$ {}^{B}\mathbf{v}_{A,B}^{\wedge} = {}^{A}\mathbf{H}_{B}^{-1}\, {}^{A}\dot{\mathbf{H}}_{B} \in \mathfrak{se}(3). \tag{2.6} $$

As a vector, the left trivialized velocity represents the velocity of frame B with respect to frame A expressed in frame B

$$ {}^{B}\mathbf{v}_{A,B} := \begin{bmatrix} {}^{B}\boldsymbol{v}_{A,B} \\ {}^{B}\boldsymbol{\omega}_{A,B} \end{bmatrix} \in \mathbb{R}^6, \tag{2.7} $$

where

$$^{B}\boldsymbol{v}_{A,B} := {}^{A}\mathbf{R}_{B}^{T}\,{}^{A}\dot{\mathbf{o}}_{B}, \tag{2.8}$$

$$^{B}\boldsymbol{\omega}_{A,B}^{\wedge} := {}^{A}\mathbf{R}_{B}^{T}\,{}^{A}\dot{\mathbf{R}}_{B}. \tag{2.9}$$

The right trivialized velocity of frame B with respect to A is given by the twist

$$^{A}\mathbf{v}_{A,B}^{\wedge} = {}^{A}\dot{\mathbf{H}}_{B}\,{}^{A}\mathbf{H}_{B}^{-1} \in \mathfrak{se}(3), \tag{2.10}$$

As a vector the right trivialized velocity is given by

$$^{A}\mathbf{v}_{A,B} := \begin{bmatrix} {}^{A}\boldsymbol{v}_{A,B} \\ {}^{A}\boldsymbol{\omega}_{A,B} \end{bmatrix} \in \mathbb{R}^{6}, \tag{2.11}$$

where

$$^{A}\boldsymbol{v}_{A,B} := {}^{A}\dot{\mathbf{o}}_{B} - {}^{A}\dot{\mathbf{R}}_{B}\,{}^{A}\mathbf{R}_{B}^{T}\,{}^{A}\mathbf{o}_{B}, \tag{2.12}$$

$$^{A}\boldsymbol{\omega}_{A,B}^{\wedge} := {}^{A}\dot{\mathbf{R}}_{B}\,{}^{A}\mathbf{R}_{B}^{T}. \tag{2.13}$$

In some cases the specification of a natural velocity ${}^{A}\dot{\mathbf{o}}_{B}$ and ${}^{A}\boldsymbol{\omega}_{A,B}$ is desired. In that situation a mixed frame representation can used $B[A] := (\mathbf{o}_{B},[A])$, which is a frame whose origin coincides with frame B, but has the orientation of frame A. This mixed velocity is given by

$$^{B[A]}\mathbf{v}_{A,B} = \begin{bmatrix} {}^{A}\dot{\mathbf{o}}_{B} \\ {}^{A}\boldsymbol{\omega}_{A,B} \end{bmatrix}. \tag{2.14}$$

The velocities just presented can be expressed in an arbitrary frame using the linear transformation

$$^{A}\mathbf{X}_{B} := \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{o}_{B}^{\wedge}\,{}^{A}\mathbf{R}_{B} \\ 0_{3\times 3} & {}^{A}\mathbf{R}_{B} \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{2.15}$$

For an arbitrary frame C the velocity frame transformations are given by

$$^{C}\mathbf{v}_{A,B} := {}^{C}\mathbf{X}_{A}\,{}^{A}\mathbf{v}_{A,B} = {}^{C}\mathbf{X}_{B}\,{}^{B}\mathbf{v}_{A,B}. \tag{2.16}$$

The time derivative of (2.15) is given by

$$^{A}\dot{\mathbf{X}}_{B} = {}^{A}\mathbf{X}_{B}\,{}^{B}\mathbf{v}_{A,B}\times, \tag{2.17}$$

where the term ${}^{B}\mathbf{v}_{A,B}\times$ is defined as

$$^{B}\mathbf{v}_{A,B}\times := \begin{bmatrix} {}^{B}\boldsymbol{\omega}_{A,B}^{\wedge} & {}^{B}\boldsymbol{v}_{A,B}^{\wedge} \\ 0_{3\times 3} & {}^{B}\boldsymbol{\omega}_{A,B}^{\wedge} \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{2.18}$$

Equation (2.18) is referred to as the matrix representation of the cross product on $\mathbb{R}^{6}$. The cross product of velocity vectors satisfies the distributive property

$$^{A}\mathbf{X}_{B}\,{}^{B}\mathbf{v}_{A,B}\times = ({}^{A}\mathbf{X}_{B}\,{}^{B}\mathbf{v}_{A,B}) \times {}^{A}\mathbf{X}_{B} = {}^{A}\mathbf{v}_{A,B} \times {}^{A}\mathbf{X}_{B}. \tag{2.19}$$

2.1.3 Acceleration Vectors

The acceleration vectors associated with the left, right trivialized and mixed velocity are its time derivatives, defined as

$$ {}^{C}\dot{\mathbf{v}}_{A,B} := \frac{d}{dt}({}^{C}\mathbf{v}_{A,B}) = \begin{bmatrix} {}^{C}\dot{\boldsymbol{v}}_{A,B} \\ {}^{C}\dot{\boldsymbol{\omega}}_{A,B} \end{bmatrix} \in \mathbb{R}^6. \tag{2.20} $$

So that the left and right trivialized accelerations are given by ${}^{B}\dot{\mathbf{v}}_{A,B}$ and ${}^{A}\dot{\mathbf{v}}_{A,B}$, respectively. The left and right trivialized accelerations are related by the transformation

$$ {}^{A}\dot{\mathbf{v}}_{A,B} = {}^{A}\mathbf{X}_{B}\,{}^{B}\dot{\mathbf{v}}_{A,B}. \tag{2.21} $$

The mixed frame acceleration is given by

$$ {}^{B[A]}\dot{\mathbf{v}}_{A,B} = \begin{bmatrix} {}^{B[A]}\dot{\boldsymbol{v}}_{A,B} \\ {}^{B[A]}\dot{\boldsymbol{\omega}}_{A,B} \end{bmatrix} = \begin{bmatrix} {}^{A}\ddot{\mathbf{o}}_{B} \\ {}^{A}\dot{\boldsymbol{\omega}}_{A,B} \end{bmatrix} \in \mathbb{R}^6. \tag{2.22} $$

2.1.4 Forward Kinematics

Robots often consist out of multiple links and joints and one would like to be able to express the pose of an end-effector as a function of its joint states, referred to as forward kinematics (FK). In this thesis only fixed-base serial manipulators with n joints are considered, for example, the three joint manipulator in Figure 2.2.

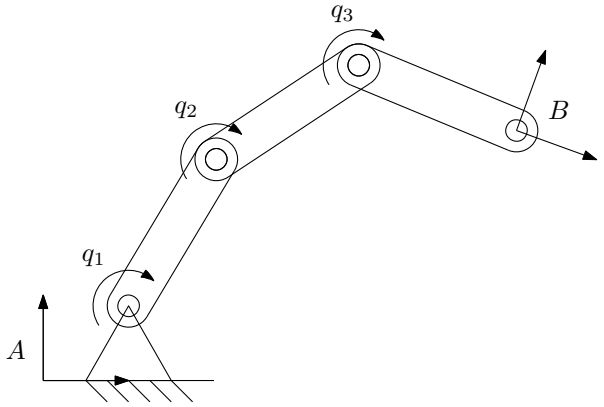


Figure 2.2: *Illustration of a fixed-base three joint manipulator, the joint states are denoted by q_i. The pose of the end-effector can be described by the homogeneous transformation matrix ${}^{A}\mathbf{H}_{B}$, relating frame B to the inertial frame A.*

The joint states are collected in the vector $\boldsymbol{q} \in \mathbb{R}^n$. The pose of the end-effector frame B expressed in frame A can be described as a FK function of the joint states

$$ ({}^{A}\mathbf{o}_{B}, {}^{A}\mathbf{R}_{B}) = \boldsymbol{f}(\boldsymbol{q}). \tag{2.23} $$

This function can be found as a composition of homogeneous transformation matrices relating each joint to the next. The velocity of the end-effector frame B with respect to frame A expressed in frame A is related to the time derivative of the joint states $\dot{\boldsymbol{q}}$ by means of the Jacobian ${}^{A}\boldsymbol{J}_{A,B}$

$$ {}^{A}\mathbf{v}_{A,B} = {}^{A}\boldsymbol{J}_{A,B}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{2.24} $$

Taking the time derivative of (2.24), results in the forward kinematics on acceleration level. The acceleration of the end-effector frame B with respect to frame A expressed in frame A as a function of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ is given by

$$^{A}\dot{\mathbf{v}}_{A,B} = {}^{A}\boldsymbol{J}_{A,B}(\boldsymbol{q})\ddot{\boldsymbol{q}} + {}^{A}\dot{\boldsymbol{J}}_{A,B}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}. \tag{2.25}$$

2.2 Optimization-Based Control

As discussed in Chapter 1 the objective is to compare an SE(3) and task-point reach task in the context of quadratic programming (QP) robot control. This section elaborates on the QP controller used for both reach tasks.

A general QP problem consists of a quadratic cost function with linear equality and inequality constraints is given by [20, Chapter 16]

$$\begin{aligned} \min_{\boldsymbol{\chi}} \quad & \frac{1}{2}\boldsymbol{\chi}^T G\boldsymbol{\chi} + g^T\boldsymbol{\chi}. \\ \text{s.t.} \quad & C_E\boldsymbol{\chi} = c_E \\ & C_I\boldsymbol{\chi} \leq c_I, \end{aligned} \tag{2.26}$$

where $\boldsymbol{\chi} \in \mathbb{R}^n$ is the variable used to minimize the cost function, $G \in \mathbb{R}^{n\times n}$ is a positive definite matrix, $g \in \mathbb{R}^n$ a vector, $C_E \in \mathbb{R}^{m\times n}$ and $c_E \in \mathbb{R}^m$ form the equality constraints and $C_I \in \mathbb{R}^{m\times n}$ and $c_I \in \mathbb{R}^m$ the inequality linear constraints. The cost function of (2.26) is often written in a different form by using $G = A^TA$ and $g = -A^Tb$, which results in

$$\min_{\boldsymbol{\chi}} \quad \frac{1}{2}||A\boldsymbol{\chi} - b||^2. \tag{2.27}$$

2.2.1 Task-Based Constrained Optimal Control

The current and desired pose used in a reach task are specified in task-space, the term task-based control is therefore used to refer to a control strategy that uses feedback on the current state of the robot in task-space. Task-based control is the minimization of a task error under the constraint of the kinematics of the robot which can be realized in a QP formulation. The resulting output of the controller are the joint states, therefore effectively solving the inverse kinematics (IK) problem. QP problems require linear constraints and the kinematics of the robot are linear in $\dot{\boldsymbol{q}}$ or $\ddot{\boldsymbol{q}}$ when considered on velocity (2.24) or acceleration (2.25) level respectively. For future considerations, an acceleration based formulation is preferred if the robot dynamics are to be taken into account, since the robot dynamics of a fixed-base serial manipulator are linear in $\ddot{\boldsymbol{q}}$, while being non-linear in $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Since the output of the QP controller allows for jumps in the optimization variable, another benefit of defining the QP controller at acceleration level is that the resulting motion of the robot is smoother compared to a QP controller at velocity level.

The exact mathematical form of the kinematic constraint depends on the representation of the pose and therefore the kinematic constraint on acceleration level is left unambiguous, until the SE(3) and task-point reach tasks are formally introduced in Chapter 3. A general task-based

QP formulation on acceleration level, expressed in frame A, is then written as

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}}} \quad & \frac{1}{2}||{}^{A}\boldsymbol{E}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})||^2_W \\ \text{s.t.} \quad & {}^{A}\boldsymbol{a} = {}^{A}\boldsymbol{J}\ddot{\boldsymbol{q}} + {}^{A}\dot{\boldsymbol{J}}\dot{\boldsymbol{q}} \\ & C_E\ddot{\boldsymbol{q}} = c_E \\ & C_I\ddot{\boldsymbol{q}} \leq c_I, \end{aligned} \tag{2.28}$$

where ${}^{A}\boldsymbol{E}$ is the task error on acceleration level, ${}^{A}\boldsymbol{a}$ the acceleration of a part of the robot depending on the representation used for its pose, ${}^{A}\boldsymbol{J}$ the Jacobian, also dependent on the representation of the pose, and with $||\cdot||_W$ the weighted norm

$$||\boldsymbol{x}||_W = \sqrt{\boldsymbol{x}^T W \boldsymbol{x}} \in \mathbb{R}. \tag{2.29}$$

The task error ${}^{A}\boldsymbol{E}$ should represent stabilized error dynamics on acceleration level in order to evoke stability of the QP controller in case the minimization is successful. Stable error dynamics are derived for control on $\mathbb{R}^3$ and on SE(3) in the next two sections.

2.2.2 Feedback and Feedforward Control on $\mathbb{R}^3$

Given a point $\boldsymbol{p}$ with the reference point $\boldsymbol{d}$, which can be a function of time. The task error is defined as the error between the two vector quantities

$${}^{A}\boldsymbol{e}_p = {}^{A}\boldsymbol{p} - {}^{A}\boldsymbol{d}, \tag{2.30}$$

The system to be controlled are the kinematics of a point on acceleration level expressed in the inertial frame A.

$${}^{A}\ddot{\boldsymbol{p}} = \boldsymbol{u} \tag{2.31}$$

with $\boldsymbol{u}$ the controller action/output. To create a stabilizing controller that drives the task error ${}^{A}\boldsymbol{e}_p$ to zero a PD-controller with feedforward is used

$$\boldsymbol{u} = -K_p{}^{A}\boldsymbol{e}_p - K_d{}^{A}\dot{\boldsymbol{e}}_p + {}^{A}\ddot{\boldsymbol{d}}. \tag{2.32}$$

Applying this controller to (2.31) leads to the error dynamics

$${}^{A}\ddot{\boldsymbol{e}}_p + K_d{}^{A}\dot{\boldsymbol{e}}_p + K_p{}^{A}\boldsymbol{e}_p = 0. \tag{2.33}$$

Critically damped behaviour can be achieved for $K_d = 2\sqrt{K_p}$. Equation (2.33) can be used as a cost function in (2.28).

2.2.3 PD Control on SE(3)

A PD controller that almost globally stabilizes the origin of SE(3) is presented in [11]. This section aims to use this result to derive a cost function on acceleration level such that is compatible with the QP control formulation in 2.28. The derivation follows the same approach as in [21]. The PD controller of [11] is given by

$$\dot{\boldsymbol{E}} = \mathbf{v}_E^{\wedge}\boldsymbol{E} \tag{2.34}$$

$$\dot{\mathbf{v}}_E = -\boldsymbol{K}_d\mathbf{v}_E - \boldsymbol{K}_p(\log(\boldsymbol{E}))^{\vee}, \tag{2.35}$$

where $\boldsymbol{E} \in \mathrm{SE}(3)$ and $\mathbf{v}_E^\wedge \in \mathbb{R}^{4\times4}$. The pose error $\boldsymbol{E}$ is defined as the homogeneous transformation matrix from the desired frame D to the current body frame B, so

$$\boldsymbol{E} := {}^A\mathbf{H}_B^{-1\,A}\mathbf{H}_D \tag{2.36}$$

The twist $\mathbf{v}_E$ denotes the twist of the desired frame with respect to the current frame, expressed in the current frame, such that

$$\mathbf{v}_E = {}^B\mathbf{v}_{B,D}. \tag{2.37}$$

For the sake of clarity, let $D := {}^A\mathbf{H}_D$, $B := {}^A\mathbf{H}_B$, $E := B^{-1}D$, $\mathbf{v}_D = {}^A\dot{\mathbf{v}}_{A,D}$, $\mathbf{v}_B = {}^A\dot{\mathbf{v}}_{A,B}$ and $\boldsymbol{X}_B = {}^A\boldsymbol{X}_B$. From (2.34), it follows that

$$\begin{aligned}
(\dot{\boldsymbol{B}^{-1}})\boldsymbol{D} + \boldsymbol{B}^{-1}\dot{\boldsymbol{D}} &= \mathbf{v}_E^\wedge \boldsymbol{B}^{-1}\boldsymbol{D},\\
(\dot{\boldsymbol{B}^{-1}})\boldsymbol{B} + \boldsymbol{B}^{-1}\dot{\boldsymbol{D}}\boldsymbol{D}^{-1}\boldsymbol{B} &= \mathbf{v}_E^\wedge,\\
-\boldsymbol{B}^{-1}\dot{\boldsymbol{B}} + \boldsymbol{B}^{-1}\dot{\boldsymbol{D}}\boldsymbol{D}^{-1}\boldsymbol{B} &= \mathbf{v}_E^\wedge,\\
-\boldsymbol{B}^{-1}\mathbf{v}_B^\wedge\boldsymbol{B} + \boldsymbol{B}^{-1}\mathbf{v}_D^\wedge\boldsymbol{B} &= \mathbf{v}_E^\wedge,\\
\boldsymbol{B}^{-1}(\mathbf{v}_D^\wedge - \mathbf{v}_B^\wedge)\boldsymbol{B} &= \mathbf{v}_E^\wedge.
\end{aligned} \tag{2.38}$$

Where was used that $(\dot{\boldsymbol{B}^{-1}}) = -\boldsymbol{B}^{-1}\dot{\boldsymbol{B}}\boldsymbol{B}^{-1}$, $\mathbf{v}_D^\wedge = \dot{\boldsymbol{D}}\boldsymbol{D}^{-1}$ and $\mathbf{v}_B^\wedge = \dot{\boldsymbol{B}}\boldsymbol{B}^{-1}$. This equation can be transformed from skew-symmetric twist velocity matrices to twist velocity vectors using $\mathbf{v} = (\mathbf{v}^\wedge)^\vee$, which results in

$$\mathbf{v}_E = \boldsymbol{X}_B^{-1}(\mathbf{v}_D - \mathbf{v}_B) \tag{2.39}$$

Where the following insight was used

$${}^B\mathbf{v}_{A,B}^\wedge = {}^A\mathbf{H}_B^{-1\,A}\mathbf{v}_{A,B}^\wedge\,{}^A\mathbf{H}_B \in \mathfrak{se}(3) \tag{2.40}$$

$${}^B\mathbf{v}_{A,B} = {}^A\boldsymbol{X}_B^{-1\,A}\mathbf{v}_{A,B} \in \mathbb{R}^6. \tag{2.41}$$

The error dynamics can then be written on acceleration level by taking the time derivative of (2.39)

$$\begin{aligned}
\dot{\mathbf{v}}_E &= \dot{\boldsymbol{X}}_B^{-1}(\mathbf{v}_D - \mathbf{v}_B) + \boldsymbol{X}_B^{-1}(\dot{\mathbf{v}}_D - \dot{\mathbf{v}}_B)\\
\dot{\mathbf{v}}_E &= -\boldsymbol{X}_B^{-1}\mathbf{v}_B \times (\mathbf{v}_D - \mathbf{v}_B) + \boldsymbol{X}_B^{-1}(\dot{\mathbf{v}}_D - \dot{\mathbf{v}}_B)\\
\boldsymbol{X}_B\dot{\mathbf{v}}_E &= -\mathbf{v}_B \times \mathbf{v}_D + \dot{\mathbf{v}}_D - \dot{\mathbf{v}}_B.
\end{aligned} \tag{2.42}$$

Where was used that the time derivative of ${}^A\boldsymbol{X}_B^{-1}$ can be written as

$$\begin{aligned}
(\dot{{}^A\boldsymbol{X}_B^{-1}}) &= -{}^A\boldsymbol{X}_B^{-1\,A}\dot{\boldsymbol{X}}_B\,{}^A\boldsymbol{X}_B^{-1}\\
(\dot{{}^A\boldsymbol{X}_B^{-1}}) &= -{}^A\boldsymbol{X}_B^{-1\,A}\boldsymbol{X}_B\,{}^B\mathbf{v}_{A,B} \times {}^A\boldsymbol{X}_B^{-1}\\
(\dot{{}^A\boldsymbol{X}_B^{-1}}) &= -{}^A\boldsymbol{X}_B^{-1\,A}\mathbf{v}_{A,B} \times {}^A\boldsymbol{X}_B\,{}^A\boldsymbol{X}_B^{-1}\\
(\dot{{}^A\boldsymbol{X}_B^{-1}}) &= -{}^A\boldsymbol{X}_B^{-1\,A}\mathbf{v}_{A,B}\ \times \boldsymbol{I}_{6\times6}
\end{aligned} \tag{2.43}$$

Equation (2.42) can be rewritten as

$$\dot{\mathbf{v}}_D = \dot{\mathbf{v}}_B + \mathbf{v}_B \times \mathbf{v}_D + \boldsymbol{X}_B\dot{\mathbf{v}}_E \tag{2.44}$$

Implementing the control law (2.35) gives

$$\dot{\mathbf{v}}_D = \dot{\mathbf{v}}_B + \mathbf{v}_B \times \mathbf{v}_D + \boldsymbol{X}_B(-\boldsymbol{K}_d\mathbf{v}_E - \boldsymbol{K}_p(\log(\boldsymbol{E}))^\vee), \tag{2.45}$$

which, by substitution of (2.39), is equal to

$$\dot{\mathbf{v}}_D = \dot{\mathbf{v}}_B + \mathbf{v}_B \times \mathbf{v}_D - \boldsymbol{X}_B\boldsymbol{K}_d\boldsymbol{X}_B^{-1}(\mathbf{v}_D - \mathbf{v}_B) - \boldsymbol{X}_B\boldsymbol{K}_p\boldsymbol{X}_B^{-1}\boldsymbol{X}_B(\log(\boldsymbol{E}))^\vee. \tag{2.46}$$

This equation hides the effect of the gain matrices $\boldsymbol{K}_d$ and $\boldsymbol{K}_p$ on the position and velocity error among cross terms as a result from the matrix multiplications. If the gain matrices $\boldsymbol{K}_d$ and $\boldsymbol{K}_p$ are scalar, so $\boldsymbol{K}_d = k_d$ and $\boldsymbol{K}_p = k_p$, then

$$\boldsymbol{X}_B\boldsymbol{K}_d\boldsymbol{X}_B^{-1} = k_d \tag{2.47}$$

$$\boldsymbol{X}_B\boldsymbol{K}_p\boldsymbol{X}_B^{-1} = k_p \tag{2.48}$$

such that (2.46) equals

$$\dot{\mathbf{v}}_D = \dot{\mathbf{v}}_B + \mathbf{v}_B \times \mathbf{v}_D - k_d(\mathbf{v}_D - \mathbf{v}_B) - k_p\boldsymbol{X}_B(\log(\boldsymbol{E}))^\vee. \tag{2.49}$$

This equation can be used as the cost function in 2.28 to create a reach task.

2.3 Software

This section introduces the software architecture and its components that are used for the kinematic reach task simulations in Chapter 4.

2.3.1 Robotic Operating System

The robotic operating system (ROS) is a software framework for interprocess communication [22]. There is also exists a rich set robotic related software libraries that make use of the ROS framework. ROS views a process as a node, which can subscribe to topics that receive messages from its publishers. Two nodes can therefore communicate via this publisher and subscriber model. ROS provides a 3D visualization tool called rviz which is not only useful for the visualization of a robot, it also provides debugging in the form of graphs of the robot states and user interaction with a robot in simulation. ROS also provides a tool for recording and playing back ROS topics called rosbags. This feature can be used to play back an entire simulation, partially or even used as a way to save the time evolution of the states of the robot or any other variable.

The ROS framework is used to create a controller node that contains all the QP controller related computations and a visualization node that makes use of rviz. The overall architecture is given by Figure 2.3. The following section goes into detail about the controller node.

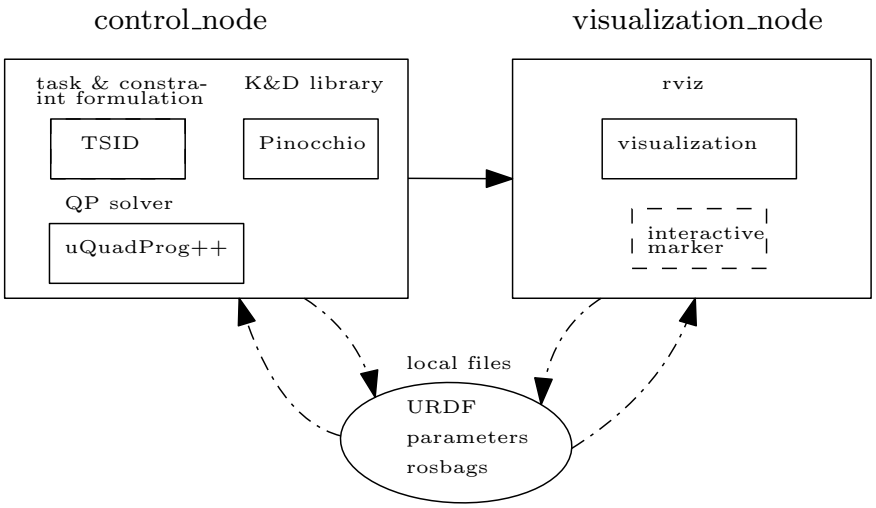


Figure 2.3: Illustration of the software architecture. The dashed lines represent optional functionality. The interprocess (node) communication is handled by ROS.

The local files represent all the files that contain parameters that configure the two nodes or stores information from both nodes in rosbags. One in particular, the unified robot description format (URDF), provides a description for an entire robot [23]. Most importantly it describes how each link is connected each joint. The QP controller software described in the next section and rviz are able to parse the URDF and use this information for kinematic calculations and 3D visualization respectively.

2.3.2 Control Node

The control node contains three main components: the task-space inverse dynamics (TSID) library [24], the kinematics and dynamics library Pinocchio [25] and a TSID modified version of QP solver uQuadProg++ [26, 27]. The TSID library provides a layer of abstraction in the creation of a QP problem by providing an interface for specifying tasks and constraints, which TSID is then able to convert into a QP problem. However, not all tasks that were advertised on its wikipage have been implemented. The relevant tasks that were implemented during the work of this thesis:

- **Joint posture task.** This task is used to move the joint angles towards a possibly time dependent reference trajectory in joint space by employing a PD controller on the joint angles.
- **SE(3) task.** This task is used to move a part of the robot to a possibly time dependent reference pose in Cartesian space by employing a PD controller on SE(3).

The use of multiple tasks can be handled in multiple different ways as discussed in Section 1.2.3. The documentation of TSID suggested that the library offers a hard hierarchy, however, currently only offers a soft hierarchy using weights. TSID makes use of the Pinocchio library for its kinematic and dynamic calculations, the exact algorithms used to calculate these have not been investigated. Finally, the QP problem created by TSID using Pinocchio is solved using a modified version of the numerical solver uQuadProg++ based on Eigen data structures [26, 27]. TSID also offers the option to choose a solver, for example qpOASES, however it seems that it has not

been implemented yet. The solver also offers an interface to monitor the state the QP solver as: unknown, optimal, infeasible, unbounded, max_iter_reached and error. A monitor like this is useful tool in debugging a library that is build around the use of QP problems.

Chapter 3

Reach Task Formulations

In this chapter, the mathematical descriptions of the two reach task formulations, SE(3) and task-point based, are formalized. In order to achieve this, the definition of a reach task and reach task function are stated first. After which, the concept of a redundant reach task is introduced together with a few examples. Then, the QP control formulation, in which the SE(3) and task-point reach tasks functions are formalized, is recalled. Finally, both reach tasks functions are formalized and compared on their ability to specify (redundant) reach tasks and the effect of their tuneable parameters on the resulting motion.

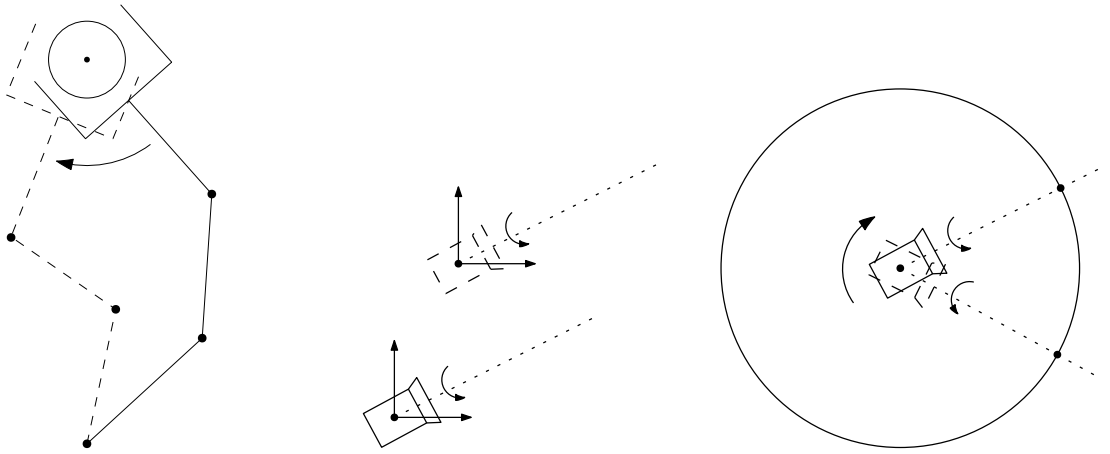
3.1 Reach Task Definition

In the context of robot control, a reach task is defined as the task to reach a certain desired position and/or orientation by a part of the robot. The associated reach task function is the mathematical formulation of the reach task in terms of a chosen error to quantify the difference between the desired and current pose of a part of the robot, the latter being a function of the robot state. Reach tasks are a fundamental task and many more complicated tasks can be build from this it. For example, in the case of a service robot: opening a door, grabbing a cup from a table and storing the groceries. The reach tasks in these examples, all require the specification of a desired position and/or orientation of the end-effector in one or more stages of the task.

3.1.1 Redundant Reach Tasks

A coordinate frame in $\mathbb{R}^3$ attached to a link of the robot has six degrees of freedom (DoF), three for position and three for orientation. A reach task specifies a desired position and/or orientation for this frame, however not every reach task requires all six DoF to be specified. As a consequence, at the goal position and/or orientation, the frame is still allowed to move in the direction of the unspecified DoF or allowed to rotate about the axis of the unspecified DoF, depending on which DoF was left unspecified. An $m \in \mathbb{R}^3$ DoF reach task is said to have m DoF specified by the desired position and/or orientation of the frame attached to a link of the robot. This is equal to imposing m soft constraints on the pose of this frame (i.e. the dimension of the reach task function). The class of reach tasks that specify $m < 6$ DoF are referred to as redundant reach tasks. For a coordinate frame in $\mathbb{R}^2$, a reach task that specifies $m < 3$ DoF is a redundant reach task.

Redundant reach tasks are common for robotic systems that use a manipulator to grab objects or a camera to look at objects. Figure 3.1 illustrates three examples.



(a) *cylinder grasping* (b) *direction task* (c) *inspect inner circle from center*

Figure 3.1: *Redundant reach task examples. Illustration (a) shows a 2D example, illustrations (b) and (c) show 3D examples. The arrows indicate the DoF that have been left unspecified and the dashed version then shows a possible new pose that still complies with the requested reach task.*

The first example in Figure 3.1a shows a three DoF robot arm where the reach task is to position the gripper before grabbing the object. This particular example is actually a 2D problem, but it has a 3D equivalent. For the task at hand the rotation about the axis of the object is left free (the center point) and therefore specifies two DoF out of the three available DoF. The second example in Figure 3.1b shows a camera and its reach task is look in a certain direction (i.e. a vector direction task). For this task the rotation about the dashed axis and the movement in all three directions are not important and therefore only specifies two DoF out of the six DoF. The third example in Figure 3.1c, again shows a camera and its reach task is to position itself in the center of the circle and always point towards the inner edge of the circle. For this task the rotation about the center of the circle and around the pointing direction are irrelevant. This means that the three position DoF and one orientation DoF are specified. However, notice that the body can still obtain any orientation by successive rotations about the two unspecified DoF. This leads to the notion that this reach task only specifies four DoF instantaneously. Meaning, that at each instance of time the camera is specified by four DoF, however, in time it can reach poses these four DoF would not allow instantaneously. Another example of a three DoF position + one DoF orientation reach task can be found in [28].

From these examples it becomes clear that it is important to, firstly, realize that these tasks are redundant by nature, and secondly, to design reach tasks that do not specify more DoF than necessary. As a result, the robot has a number of free DoF that could be used for additional tasks, for example, avoiding an obstacle. This is related to the concept of redundancy introduced in Section 2.2. If a robot with n DoF is to execute a task of m DoF, a task redundancy exists when $n > m$ with $n - m$ the task redundancy degree. The difference is that task redundancy considers the DoF of the robot as a whole, whereas, redundant reach tasks consider the DoF of a frame attached to a link of the robot, most often the end-effector of a robot arm.

3.2 Embedding Reach Tasks in the QP Formulation

As mentioned in Section 1.3, the focus of this thesis is on comparing the SE(3) and task-point reach task formulations that can handle task redundancy and multiple tasks simultaneously. In Section 2.2, a task-based QP formulation was developed that is able to take each of these elements into consideration. The formulation achieves this by viewing the reach task as an optimal control problem in which constraints, such as the joint limits, can be added as constraints in the QP problem. As any control problem, a task error needs to be defined and control law that aims to minimize this error. The QP controller minimizes the task error at acceleration level, where the joint accelerations $\ddot{\boldsymbol{q}}$ are the optimization variables. The control laws that drive the task error to zero, should therefore also output accelerations in order to add them to the QP formulation.

In the following two sections the representation of the goal, task error, and a controller on acceleration level that aims to minimize this error for the SE(3) and task-point reach tasks respectively. Both controllers are then integrated in the aforementioned QP formulation.

3.3 Reach Task Formulation on SE(3)

The homogeneous transformation matrix ${}^{A}\mathbf{H}_{B} \in \mathrm{SE}(3)$ is used to represent the pose (position and orientation) of a frame B relative to a frame A. Consider the six DoF fixed-base serial manipulator, shown in Figure 3.2.

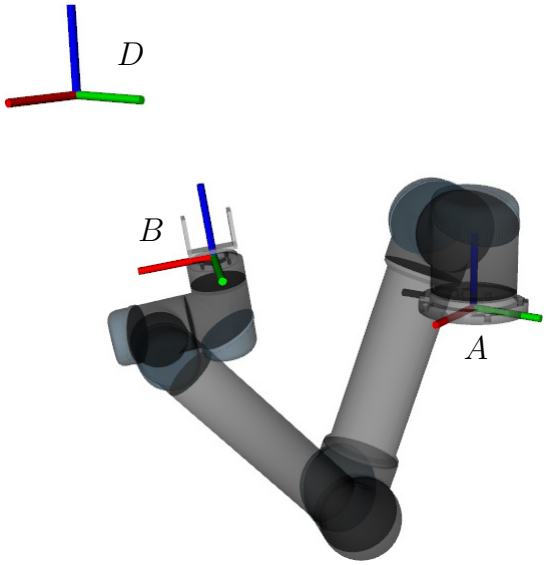


Figure 3.2: *Visualization of the SE(3) reach task for a serial manipulator. The goal of this task is to match frame B with reference frame D. Frame A is the inertial frame.*

The SE(3) reach task is then to match the link-fixed frame B with the reference frame D (possibly, a function of time), where frame A is the inertial frame. Mathematically, this means that ${}^{A}\mathbf{H}_{B}$ should become equal to ${}^{A}\mathbf{H}_{D}$. If the reference frame D is a function of time, then the twist ${}^{A}\mathbf{v}_{A,D} \in \mathbb{R}^6$ expresses the reference velocity of frame D with respect to A expressed in A and the acceleration ${}^{A}\dot{\mathbf{v}}_{A,D} \in \mathbb{R}^6$ the reference acceleration of frame D with respect to A

expressed in A. Using the cost function (2.49) derived in Section 2.2.3 results in the SE(3) reach task formulation

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}} \in \mathbb{R}^n} \quad & \frac{1}{2} || {}^A\dot{\mathbf{v}}_{A,B} - {}^A\dot{\mathbf{v}}^*_{A,D} ||^2_W \\ \text{s.t.} \quad & {}^A\dot{\mathbf{v}}_{A,B} = {}^A\boldsymbol{J}_{A,B}\ddot{\boldsymbol{q}} + {}^A\dot{\boldsymbol{J}}_{A,B}\dot{\boldsymbol{q}} \\ & {}^A\dot{\mathbf{v}}^*_{A,D} = k_p {}^A\boldsymbol{X}_B (\log({}^A\mathbf{H}_B^{-1\,A}\mathbf{H}_D))^\vee + k_d({}^A\mathbf{v}_{A,D} - {}^A\mathbf{v}_{A,B}) \\ & \qquad + {}^A\dot{\mathbf{v}}_{A,D} - {}^A\mathbf{v}_{A,B} \times {}^A\mathbf{v}_{A,D}, \end{aligned} \tag{3.1}$$

where n is the number of joints, $W \in \mathbb{R}^{6\times6}$ the weighting matrix, $\boldsymbol{q} \in \mathbb{R}^n$ the joint angle vector, ${}^A\mathbf{v}_{A,B} \in \mathbb{R}^6$ the twist expressing the velocity of frame B relative to A expressed in A, ${}^A\dot{\mathbf{v}}_{A,B} \in \mathbb{R}^6$ the acceleration of frame B relative to A expressed in A, ${}^A\boldsymbol{J}_{A,B} \in \mathbb{R}^{6\times n}$ the Jacobian relating the velocity of frame B with respect to A expressed in A and k_p and $k_d \in \mathbb{R}$ the proportional and derivative gain that are used to control the speed of convergence of B to D. This reach task formulation represents the minimization of the error dynamics on acceleration level (second order system on SE(3)) and almost globally stabilizes the origin of SE(3). This minimization is equivalent to executing an SE(3) reach task.

3.3.1 Specification of (Redundant) Reach Tasks Using the SE(3) Reach Task Formulation

The SE(3) reach task function cannot be used to create redundant reach tasks since the reference homogeneous transformation matrix ${}^A\mathbf{H}_D$ cannot be split into smaller components without also changing the control law and hence the reach task formulation, as shown in the literature review of Section 1.2.

3.4 Task-Point Reach Task Formulation

Consider Figure 3.3, where a six DoF manipulator is used to demonstrate the basic idea of a task-point based reach task.

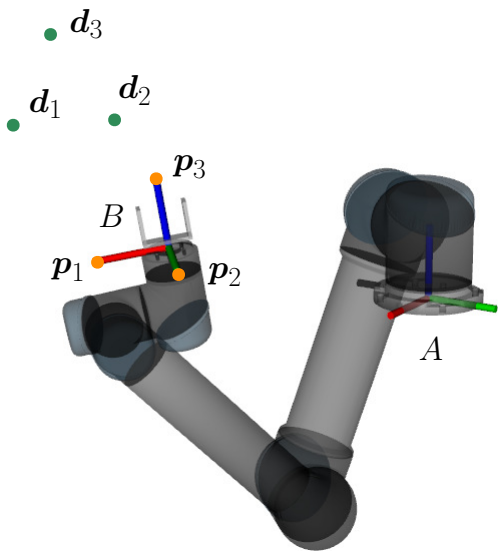


Figure 3.3: *Illustration of the task-point reach task for a six DoF serial manipulator. The goal of this task is to match the body-fixed points $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ with corresponding reference points $\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3$. Frame B is the end-effector frame and frame A is the inertial frame.*

The objective of a task-point reach task is to match a set of points $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N$ that are rigidly attached to a link of the robot with the corresponding reference points $\boldsymbol{d}_1, \cdots, \boldsymbol{d}_N$ in $\mathbb{R}^3$, which can be a function of time. In the example of Figure 3.3, three points that are rigidly attached to the end-effector link have been chosen and placed on each of the axis of the end-effector frame B. Note that these points do not have to lie on frame B or the link necessarily, just rigidly attached to the link. Intuitively, one can already tell that these reference points indirectly specify the position and orientation of the end-effector. A certain set of points can be used to create a fully specified reach task and removing some points then leads to redundant reach tasks. The construction of these reach tasks is discussed in Section 3.4.1.

In order to create a motion of each point $\boldsymbol{p}_i$ towards its reference point $\boldsymbol{d}_i$ the coordinates of these points need to be quantified and a controller that drives the error between the coordinates of these points to zero to be defined. In Section 2.2.2, a feedback with feedforward controller on $\mathbb{R}^3$ (2.32) was introduced. This controller uses the error vector ${}^A\boldsymbol{e}_i = {}^A\boldsymbol{p}_i - {}^A\boldsymbol{d}_i$ as the task error for each point. It is possible to use this controller on each point separately and combine them in a QP formulation as the summation of multiple point reach tasks functions. This results in a QP controller and is referred to as the task-point reach task formulation.

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}} \in \mathbb{R}^n} \quad & \frac{1}{2} \sum_{i=1}^{N} || {}^A\ddot{\boldsymbol{p}}_i - {}^A\ddot{\boldsymbol{d}}_i^* ||^2_{W_i} \\ \text{s.t.} \quad & {}^A\ddot{\boldsymbol{p}}_i = {}^A\boldsymbol{J}_i \ddot{\boldsymbol{q}} + {}^A\dot{\boldsymbol{J}}_i \dot{\boldsymbol{q}} \\ & {}^A\ddot{\boldsymbol{d}}_i^* = -K_{p_i}({}^A\boldsymbol{p}_i - {}^A\boldsymbol{d}_i) - K_{d_i}({}^A\dot{\boldsymbol{p}}_i - {}^A\dot{\boldsymbol{d}}_i) + {}^A\ddot{\boldsymbol{d}}_i, \end{aligned} \tag{3.2}$$

where $\ddot{\boldsymbol{q}}$ are the joint accelerations, $W_i \in \mathbb{R}^{3\times3}$ the weighting matrix of each task, $K_{p_i} \in \mathbb{R}^{3\times3}$ and $K_{d_i} \in \mathbb{R}^{3\times3}$ are the gain matrices for position and velocity error respectively and ${}^A\ddot{\boldsymbol{d}}_i^* \in \mathbb{R}^3$ the output of the feedback and feedforward control law. The Jacobian is defined as ${}^A\boldsymbol{J}_i := \partial {}^A\boldsymbol{p}_i(\boldsymbol{q}) / \partial \boldsymbol{q} \in \mathbb{R}^{3\times3}$, where ${}^A\boldsymbol{p}_i(\boldsymbol{q}) \in \mathbb{R}^3$ are the forward kinematics of each point $\boldsymbol{p}_i$ expressed

in frame A.

The idea of using multiple points to control the position and/or orientation of a rigid-body is inspired by the work of [18], in which the task-function approach was used for sensor-based control tasks of robot manipulators. The task-function approach can be seen as an even more general formulation.

3.4.1 Specification of (Redundant) Reach Tasks Using the Task-Point Reach Task Formulation

The task-point reach task formulation has been formalized as (3.2) and can be seen as a template from which it is possible to specify different (redundant) reach tasks. This section explores the two main parameters for specifying (redundant) reach tasks: the amount of points and the location of each of the link-fixed points $\boldsymbol{p}_i$ and reference points $\boldsymbol{d}_i$. Note that in this section the relative distances among $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N$ and among $\boldsymbol{d}_1, \cdots, \boldsymbol{d}_N$ are chosen to be equal in order to preserve predictability of the resulting motion. As for the amount of points, it is well known that three points are the minimum amount of points required to fully constrain the six DoF of a rigid-body in space. However, not any three arbitrary points have this property. The requirement is that the third point does not lie on the line spanned by the first two points. Considering the left illustration of Figure 3.4, this requirement is written as $\boldsymbol{v}_1 \times \boldsymbol{v}_2 \neq 0$, where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2 \in \mathbb{R}^3$. The three points indirectly represent the pose of a link and it is possible to extract a frame from them using e.g. Gram–Schmidt orthonormalization to quantify this pose, see the right illustration of Figure 3.4. In this case the Gram–Schmidt orthonormalization would apply to $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, which creates the 2D orthonormal frame represented by $\boldsymbol{X}_B$ and $\boldsymbol{Y}_B$. The third axis is then found by using the cross product on the two axes of this frame $\boldsymbol{Z}_B = \boldsymbol{X}_B \times \boldsymbol{Y}_B$, which creates a right handed frame with $\mathbf{o}_B = \boldsymbol{p}_1$. Hence, using three points to represent the pose of of a link of the robot is equivalent to using a frame.

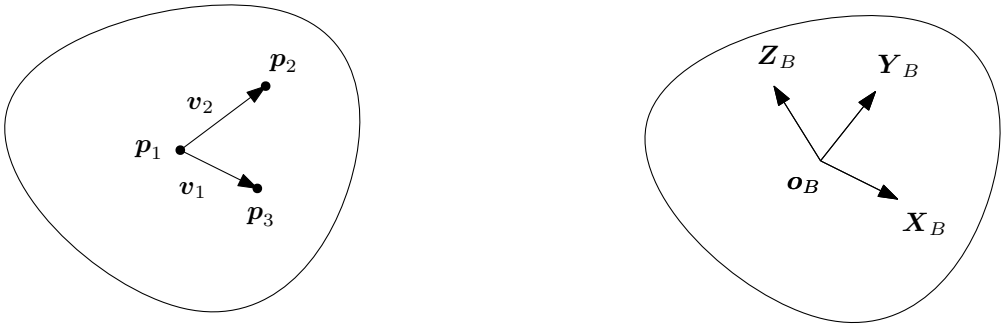


Figure 3.4: *Illustration of the extraction of a frame B from the three points $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$.*

Position Reach Task. A position reach task function is easily created from (3.2) by setting N=1 (i.e. using one point) and by specifying the reference position ${}^A\boldsymbol{d}$, which can be a function of time, for the corresponding point ${}^A\boldsymbol{p}$ that is attached to a link of the robot. The QP controller will then try to drive the task error ${}^A\boldsymbol{e} = {}^A\boldsymbol{p} - {}^A\boldsymbol{d}$ to zero by minimizing the cost function. The position reach task function just described specifies the position ${}^A\boldsymbol{d}$ in all three position DoF and leaves all three orientation DoF free. This reach task is therefore already redundant, but it is also possible to only specify two or even one position DoF, see Figure 3.5, making it even more redundant. Where the notation $\boldsymbol{d}_{x,z} \in \mathbb{R}^2$ is employed to represent a reference line specified by

only the x and z coordinate, hence leaving the y DoF unspecified. The same logic applies to $\boldsymbol{d}_z \in \mathbb{R}$, which represents a reference plane only specified by z coordinates.

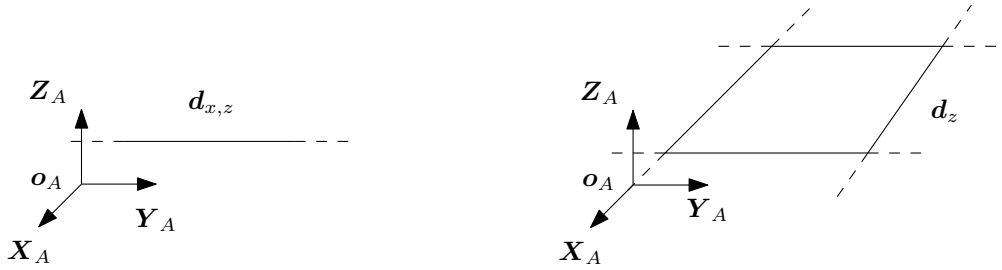


Figure 3.5: *Illustration of two redundant reference positions for a task-point reach task.*

This is achieved by not taking identity for W in (3.2), but by setting the diagonal element of the identity matrix to zero that corresponds to the DoF the reach task leaves unspecified, effectively reducing the dimension of reach task function. In the case of Figure 3.5a, the result is that the fixed point ${}^A\boldsymbol{p}$ is controlled towards the reference line ${}^A\boldsymbol{d}_{x,z}$ instead of a reference point, and is allowed to move on this line. Resulting in the cost function $\frac{1}{2}||{}^A\ddot{\boldsymbol{p}}_{x,z} - {}^A\ddot{\boldsymbol{d}}_{x,z}||^2$. In case of Figure 3.5b, the fixed point ${}^A\boldsymbol{p}$ is controlled towards the surface spanned by the two unspecified axes, and is allowed to move on this surface. Resulting in the cost function $\frac{1}{2}||{}^A\ddot{\boldsymbol{p}}_z - {}^A\ddot{\boldsymbol{d}}_z||^2$. This last case, could for example be used to position the end-effector of a robot to be a certain distance from a table (i.e. hover over the table). Note that for highly redundant reach tasks, like the two reach tasks in Figure 3.5, the optimization is made harder due to the many DoF that allow for drifting. In practical applications, these reach tasks are usually constrained (i.e. finite lines and surfaces) or used in conjunction with additional tasks.

Position + Direction Reach Task. A direction reach task specifies the direction a link needs to point at, but leaves the position of the link free, see Figure 3.1b. A position + direction reach task also specifies the location from where the link needs to point. The reach task function is created from (3.2) by setting N=2 (i.e. using two points) and by specifying the reference positions ${}^A\boldsymbol{d}_1$ and ${}^A\boldsymbol{d}_2$ for the corresponding points ${}^A\boldsymbol{p}_1$ and ${}^A\boldsymbol{p}_2$ that are attached to a link of the robot. The line through both points specifies the direction of the reach task, see Figure 3.6. This reach task specifies five DoF and only leaves the rotation about the direction vector free.

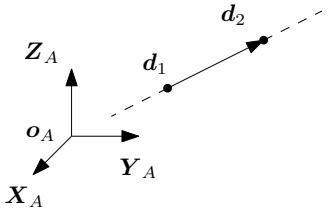


Figure 3.6: *Illustration of a task-point reach task using two reference points.*

s

Pose Reach Task. As discussed in Section 3.4.1, three points can fully describe the six DoF pose of a link. A pose reach task function is therefore created from (3.2) by setting N=3 (i.e. using three points) and by specifying the reference positions ${}^A\boldsymbol{d}_1, \cdots, {}^A\boldsymbol{d}_3$ for the corresponding

points ${}^A\boldsymbol{p}_1, \cdots, {}^A\boldsymbol{p}_3$ that are attached to a link of the robot, see for example Figure 3.3. However, a disadvantage of this reach task function is that it does not separate the position from orientation One solution could be to use two successive QP controllers. The first QP controller only controls the position by using one point (3.2) (N=1) and once this position is reached, switch to a second QP controller that controls the pose (3.2) (N=3). The second QP controller effectively only controls the orientation since the position has already been reached. However, the other way around allows for rotation during the positioning task and hence orientation is not preserved.

The choice for the location of the fixed points ${}^A\boldsymbol{p}_1, \cdots, {}^A\boldsymbol{p}_3$ also influences the relative motion between the position and orientation part of the motion. This effect is the most clearly visible when visualizing the frames that can be extracted from the points ${}^A\boldsymbol{p}_1, \cdots, {}^A\boldsymbol{p}_3$ (link frame B) and ${}^A\boldsymbol{d}_1, \cdots, {}^A\boldsymbol{d}_3$ (reference frame D). Consider the case where the origin of the link frame B already coincides with the reference frame D and only an "orientation error" around the z-axis remains, see Figure 3.7.

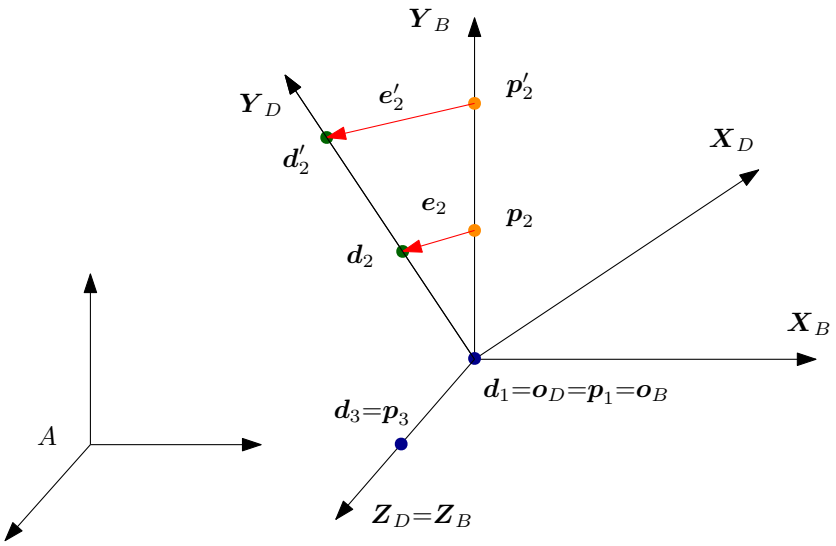


Figure 3.7: *Shows the effect of an increasing error vector $\boldsymbol{e}_2$ for the choice of the points $\boldsymbol{p}_2$ and $\boldsymbol{d}_2$ further from the origin of its frames B and D respectively. Where A is the inertial frame.*

Here the magnitude of the error ${}^A\boldsymbol{e}_2 = {}^A\boldsymbol{d}_2 - {}^A\boldsymbol{p}_2$ increases as the distance of the points from the origin increases, ${}^A\boldsymbol{e}_2' = {}^A\boldsymbol{d}_2' - {}^A\boldsymbol{p}_2' > {}^A\boldsymbol{e}_2$. In turn, this also increases the magnitude of the control action and hence affects the relative motion between position and orientation. In the case of ${}^A\boldsymbol{e}_2'$, the motion towards the implicit reference orientation is considered more important than for ${}^A\boldsymbol{e}_2$. The choice for the location of ${}^A\boldsymbol{p}_i$ and ${}^A\boldsymbol{d}_i$ can therefore be seen as a parameter that can be used to tune the resulting motion.

A method to completely separate position from orientation is proposed in the next paragraph. The separation offers greater control of the resulting motion and allows for the specification of an orientation reach task.

Orientation Reach Task. The objective of an orientation reach task is to match a link-fixed orientation frame $[B]$ with a reference orientation frame $[D]$, as illustrated in Figure 3.8. Hence,

the position of the frames in space are irrelevant.

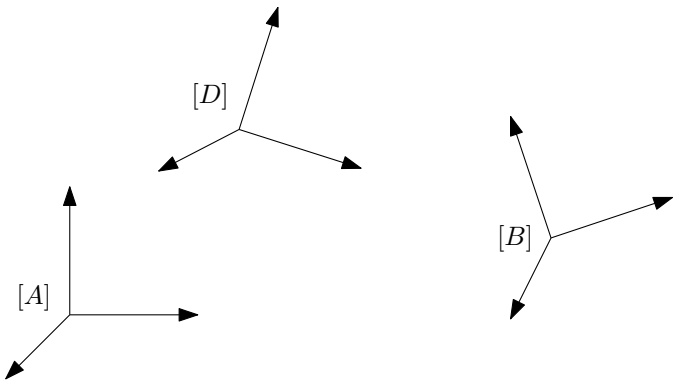


Figure 3.8: *Visualization of an orientation reach task, where the goal is to match a orientation frame* $[B]$ *with a reference orientation frame* $[D]$*. Both orientation can be related by the inertial orientation frame* $[A]$*.*

To derive an orientation reach task function from the task-point formulation (3.2) the reference points $\boldsymbol{d}_1, \cdots, \boldsymbol{d}_3$ need to be used to represent the reference orientation only and not also the position. This can be achieved by defining the reference frame D as $D := (\mathbf{o}_B, [D])$. This way, wherever frame B is, the origin of frame D coincides with it, see Figure 3.9. Note again that the points $\boldsymbol{p}_1, \cdots, \boldsymbol{p}_3$ and $\boldsymbol{d}_1, \cdots, \boldsymbol{d}_3$, do not necessarily have to lie on the coordinate axes of the frame, they only have to be sufficient to uniquely specify an orientation, which requires three points.

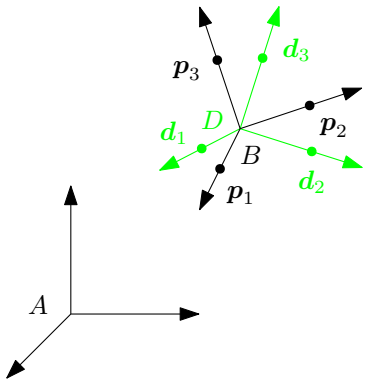


Figure 3.9: *Illustration of an orientation reach task using reference points* $\boldsymbol{d}_1, \cdots, \boldsymbol{d}_3$ *that are attached to a reference frame* D *redefined as* $D := (\mathbf{o}_B, [D])$*.*

Mathematically, this is written as

$$ {}^{A}\boldsymbol{d}_i = {}^{A}\mathbf{o}_B + {}^{A}\mathbf{R}_D \, {}^{D}\boldsymbol{d}_i. \tag{3.3} $$

However, the purpose of a task-point reach task is to only use reference points to implicitly represent reference orientation and not by means of a rotation matrix ${}^{A}\mathbf{R}_D$. So, define ${}^{D[A]}\boldsymbol{d}_i := {}^{A}\mathbf{R}_D \, {}^{D}\boldsymbol{d}_i$, which is also equal to ${}^{B[A]}\boldsymbol{d}_i$ due to the redefinition of the D frame. Equation (3.3) then becomes

$$ {}^{A}\boldsymbol{d}_i = {}^{A}\mathbf{o}_B + {}^{D[A]}\boldsymbol{d}_i. \tag{3.4} $$

The result is that the references now become ${}^{D[A]}\boldsymbol{d}_1, \cdots, {}^{D[A]}\boldsymbol{d}_3$ and with the pre-processing step of (3.4) the input to the QP controller template of (3.2) is kept intact. The change of reference ensures that the origin of the current frame B and reference frame D coincide every time step, therefore there is no position error related to the origins of these frames and also no control action that controls the position of these frames. The new reference coordinates ${}^{D[A]}\boldsymbol{d}_i$ are again intuitive, since one can now directly specify the point in the reference frame regardless of its location while still expressed in the orientation of frame A. The result is a QP controller that allows the position of the frame, attached to a link of the robot, to drift due to any cause (e.g. under the effects of gravity) while still controlling its orientation.

To complete the formulation, the pre-processing step is also derived for the velocity and acceleration references by taking the time derivative of (3.4)

$$ {}^{A}\dot{\boldsymbol{d}}_i = {}^{A}\dot{\mathbf{o}}_B + {}^{D[A]}\dot{\boldsymbol{d}}_i \tag{3.5} $$

and

$$ {}^{A}\ddot{\boldsymbol{d}}_i = {}^{A}\ddot{\mathbf{o}}_B + {}^{D[A]}\ddot{\boldsymbol{d}}_i. \tag{3.6} $$

An implication of this new formulation is that it is now possible to describe pose reach task function in two different ways. As three reference points specified as ${}^{A}\boldsymbol{d}_1, \cdots, {}^{A}\boldsymbol{d}_3$ or as three relative reference points ${}^{D[A]}\boldsymbol{d}_1, \cdots, {}^{D[A]}\boldsymbol{d}_3$ for orientation and one point ${}^{A}\boldsymbol{d}_4$ for position. Where the later method successfully separates position and orientation control and therefore gaining more control over its resulting motion.

Direction Reach Task. Before the insight of orientation separation by means of a pre-processing step of the desired points, it was not possible to specify a direction reach task, since the reference coordinates ${}^{A}\boldsymbol{d}_1$ and ${}^{A}\boldsymbol{d}_2$ also specified the position. However, by using (3.2) (N=2) and by specifying the coordinates of the reference points as ${}^{D[A]}\boldsymbol{d}_1$ and ${}^{D[A]}\boldsymbol{d}_2$, for the corresponding points ${}^{A}\boldsymbol{p}_1$ and ${}^{A}\boldsymbol{p}_2$ that are attached to a link of the robot, and using the pre-processing step of (3.4) it is possible. Which makes it possible to describe redundant reach task example of Figure 3.1b. This has the same useful implication as in the aforementioned pose task, since it is now possible create a position+direction reach task that splits the direction from the position task. Gaining more control over its resulting motion

3.5 Theoretical Comparison

This section offers a theoretical comparison between the SE(3)- and task-point-based reach task functions, formulated as (3.1) and (3.2). Both formalizations have been derived from their respective stabilized error dynamics using PD controllers and have been embedded in a task-based QP control formulation. The comparison is done on the ability of the reach task functions to specify redundant reach tasks and the effect of the tuneable parameters, present in the formulation, on the resulting motion.

3.5.1 Redundant Reach Tasks

Starting with the SE(3) reach task function (3.1). This reach task function is not able to formulate any redundant reach tasks since the reference homogeneous transformation matrix ${}^{A}\mathbf{H}_D$ cannot be split into smaller components without also changing the control law and hence

the reach task function. On the other hand, the task-point reach task function (3.2) has proven to be quite versatile in this regard. By varying the amount and the location of the reference points and link-fixed points, (3.2) was able to construct a one, two and three DoF position reach task function, a two and three DoF orientation reach task function and the combinations of these. As a result the task-point reach function now has two ways to specify a pose reach task. The first is using three points expressed in an inertial frame A. The second is using four points, three points relative to the link-fixed frame B to specify the orientation and one point expressed an inertial frame A to specify the position. The great number of (redundant) reach task the task-point reach task approach is able to create proves the effectiveness of a unified reach task formulation, which can therefore be considered successfully created.

3.5.2 Tuneable Parameters

SE(3) Reach Task. The SE(3) reach task function (3.1) has three tuneable parameters that can be used to influence the resulting motion, namely the weighting matrix W and the scalar PD gains k_p and k_d. The derivation in Section 2.2.3 showed that if the gains of the PD controller are the scalars k_p and k_d, then the effect of gains on the position and velocity error is more directly visible, instead of hiding among cross terms as a results from the matrix multiplications in (2.46). The scalar gains can be used to specify the relative importance between the position and velocity error. Eventually the resulting magnitude and direction of ${}^{A}\dot{\mathbf{v}}^{*}_{A,D}$ determines how fast and how the QP controller tries to accelerate towards its goal. The weighting matrix W, via the weighted norm (2.29), can be used to specify the relative importance and magnitude of the six components that constitute the acceleration error of the cost function in (3.1), i.e. via $\frac{1}{2}({}^{A}\dot{\mathbf{v}}_{A,B} - {}^{A}\dot{\mathbf{v}}^{*}_{A,D})^{T} W ({}^{A}\dot{\mathbf{v}}_{A,B} - {}^{A}\dot{\mathbf{v}}^{*}_{A,D}) \in \mathbb{R}$. Notice that the quadratic error of each of the six components of the error vector are added together depending on the weights between them and that therefore the weighting matrix determines the relative importance of the units of acceleration [m/s^2] and angular acceleration [rad/s^2]. Depending on the requirements of the reach task one should keep this weighing in mind, since they influence the rate of convergence of each of the six error components. The same logic applies to the previously discussed velocity error, however, since the gain is a scalar value k_d it cannot be used to signify the relative importance of the units. Due to the nature of the position error, a matrix multiplication and log function, the choice of the relative importance between the units has already been made intrinsically. A benefit of the SE(3) reach task function is that the error definition is inertial frame independent, as shown in [11, Section V].

Task-Point Reach Task. The amount of tuneable parameters, for a (redundant) task-point reach task function (3.2), varies with the amount of points used for the particular reach task. Namely, each point has three tuneable parameters, the gain matrices K_{p_i} and K_{d_i} of the PD controller and the weighting matrix W_i. The tuning of the motion using these parameters is the same as in the SE(3) reach task case, except that the weighting matrix W_i can now also be used to set the relative influence of each point on the resulting motion. And that the gain matrices K_{p_i} and K_{d_i} can be used to tune the individual components of the error. For tasks with multiple points the relative distance between the points can also be seen as an additional parameter, as was shown in Figure 3.7. This parameter can be used to tune the relative importance between the position and orientation part of e.g. a pose reach task. Notice that in the case of a task-point reach task the problem of mixing units has been removed, since the rotational component is now indirectly implemented by the use of multiple points. As discussed in the previous section, the task-point reach task function offers two ways to implement a pose

reach task. The second approach splits the position and orientation by using four points, which results in separate control over the resulting motion of the position and orientation part of the pose. The intuitive and ease of use in creating (redundant) reach task can be regarded as a benefit of this approach.

A disadvantage of the task-point reach task function is that the error definition is frame dependent. As a result, the values of the tuneable parameters are different for each frame of reference and the relation of the change in the many parameters' value to the change of reference frame is not trivial. The choice of the relative distance between points might also negatively influence the resulting motion if not chosen carefully. Take for example, the case of a pose reach task using three points, the condition on creating this pose task is that these three points do not lie on the same line. However, when does this transition occur, when do three points not lie on the same line sufficiently enough? This point of transition can be seen singularity in which a loss of controllability occurs for one rotational DoF, effectively turning it into a position+direction reach task. This question is regarded as a subject for future research.

Chapter 4

Simulations

This chapter explains how the TSID library is used to implement the task-point reach task and the challenges faced because of it. Then, by means of a simulation, a proof of concept for the task-point reach task is provided. Finally, a summary of the chapter is given.

4.1 Software Implementation

The implementation of the task-point reach task formulation (3.2) was done in the software framework presented in Section 2.3 by modifying the existing code of a SE(3) task to a 3D point reach task. The modification required the rewriting of the dimension of the Jacobian and the error definitions to match the ones in (3.2). The result is a function that can be used for each point separately. Subsequently, the cost function of each point is added to the total cost in a QP problem using weights to create a soft hierarchy. Different amount of points can be used to generate the different (redundant) task-point reach tasks as presented in Section 3.4.

Unfortunately, the process leading up to the implementation of the task-point reach task formulation was not so easy and was met with several obstacles. The first hurdle was integrating the TSID library into the ROS framework. Having never worked with ROS before, there was a lot to learn about the inner workings of both ROS and the integration of a complex C++ library. Luckily, ROS provides good documentation with many tutorials, although some tutorials were outdated. The second hurdle was understanding the TSID code in terms of what is calculated where and how? Unfortunately, the documentation of the library was outdated and claimed that the library was able to do things it was not able to, such as, a hierarchical QP (hard priority), torque tasks, joint, and actuation bounds to name the most relevant features. The code itself also barely provided comments that explain what the function calculates and what the input/output represent, so finding out what the code is calculating was hard. However, this issue was later somewhat combated by learning how to read mathematical code. Instead of trying to figure out what the code was doing the opposite stance was taken by first understanding the fundamental mathematical problem one expects the code to be solving and then trying to find these elements back in the code. This approach proved more successful in figuring out the TSID library. Pinocchio, the library used for calculating all the kinematic and dynamic equations, was also unfortunately also not well-documented. Another issue in understanding the TSID library is the fact that it makes heavy use of C++ templates, which makes the code harder to read, but also more flexible. To make the template matters even more complex it seems that the TSID library makes use of a curiously recurring template pattern (CRTP) [29] or also called F-bound

polymorphism [30] and sometimes loosely called upside-down inheritance [31], which allows a class X to derive from a class template instantiation using X itself as a template argument. The third hurdle was that TSID was hard-coded for use of floating base robots only. However, some minor adjustments to the dimensions of the vectors in TSID were made to mitigate this.

4.2 Task-Point Reach Task Proof of Concept

This section starts with the notion of singular configurations and provides a numerical simulation proof of concept of the task point-point reach task formulation presented in Section 3.4. To this end, a pose reach task represented by three points is investigated. The reach task is executed on a six DoF manipulator in the software framework introduced in Section 2.3.

4.2.1 Singular Configurations

Singular configurations are configurations where the manipulator loses one or more DoF of motion. This is caused by a rank deficiency (singularity) of the Jacobian matrix, which is configuration dependent. A review of singular configurations can be found in [3]. During the execution of test simulations using the TSID software library, singular configurations have been most often observed in configurations going from an elbow down towards an elbow up solution (or visa-versa) or when reaching for a solution outside the workspace of the manipulator (i.e. fully stretched out). Both situations go through a joint angle of 0 degrees and result in a loss of control over the velocity component parallel to the stretched out arm. The observed result is that the QP solution $\ddot{\boldsymbol{q}}^*$ goes to infinity and eventually results in a software crash caused by a NaN (not a number), an object that cannot be used for further computations. Singularities like these are currently avoided ad-hoc by starting from a certain starting configuration of the manipulator towards a certain goal position and checking if it goes through a singularity or not. As this issue is not investigated in this thesis, it presents an area of research for future work.

4.2.2 Pose Reach Task

A pose reach task is chosen as the first proof of concept simulation, because it sets up a comparison with the SE(3) reach task for future work and it is theoretically the most stable reach task on a six DoF manipulator. Specifically, the pose reach task is not redundant with respect to the manipulator DoF and hence does not leave any DoF uncontrolled. This creates a more stable foundation for the numerical solver to converge to a solution, since none of the DoF are allowed to drift. Removing the manipulator redundancy has the additional benefit of reducing the chance of drifting into a singular configuration caused by an uncontrolled DoF.

Task-Point Parameters. The QP controller of the task-point reach task formulation (3.2) is repeated here for clarity and to highlight the parameters present in the formulation. A task-point reach task formulation using three points for a six DoF manipulator is given by

$$\begin{aligned} \min_{\ddot{\boldsymbol{q}}\in\mathbb{R}^n} \quad & \frac{1}{2}\sum_{i=1}^{N} ||{}^A\ddot{\boldsymbol{p}}_i - {}^A\ddot{\boldsymbol{d}}_i^*||^2_{W_i} \\ \text{s.t.} \quad & {}^A\ddot{\boldsymbol{p}}_i = {}^A\boldsymbol{J}_i\ddot{\boldsymbol{q}} + {}^A\dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}} \\ & {}^A\ddot{\boldsymbol{d}}_i^* = -K_{p_i}({}^A\boldsymbol{p}_i - {}^A\boldsymbol{d}_i) - K_{d_i}({}^A\dot{\boldsymbol{p}}_i - {}^A\dot{\boldsymbol{d}}_i) + {}^A\ddot{\boldsymbol{d}}_i, \end{aligned} \tag{4.1}$$

with $n = 6$, $N = 3$. For this simulation the weighting matrix is chosen as $W_i = I_{3\times3}$ for every i, which means each point is chosen to be equally important. The points are chosen to be equally important, because the only goal of a proof of concept is to show the convergence of the task-point reach task formulation (4.1) and hence there is no use-case to relate requirements or specifications to. As discussed in Section 3.5, the distance between a set of points is also considered a parameter to indirectly influence the relative importance of position and orientation. In this case the three link-fixed points $\boldsymbol{p}_i$ have been chosen to lie on the tips of end-effector frame B as done in Figure 3.3 with a distance of 0.1 m, which is an arbitrary choice. The three PD gain matrices K_{p_i} and K_{d_i} are chosen to be equal to identity times the scalar $k_{p_i} = 3$ and $k_{d_i} = 2\sqrt{k_{p_i}}$ respectively. These gains have not been chosen to optimize performance, only to realize stable critically damped behavior of each point separately. The QP controller node of Figure 2.3 runs at 100 Hz in the ROS framework, hence the time-step of the QP controller is 0.01 s. As discussed in Section 2.3, a modified version of uQuadProg++ is used to solve the QP problem every time-step.

Initial Conditions & Reference Values. The starting configuration of the robot is chosen as

$$\boldsymbol{q}^{\text{init}} = \frac{1}{2}\pi[1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T, \tag{4.2}$$

to make sure it does not start in a singular configuration. The joint velocities and accelerations are set to zero, $\dot{\boldsymbol{q}}^{\text{init}} = 0_{6\times1}$ and $\ddot{\boldsymbol{q}}^{\text{init}} = 0_{6\times1}$. As discussed in the previous paragraph, the end-effector-fixed points $\boldsymbol{p}_i$ each lie on 0.1 m distance from the origin of the end-effector frame ${}^A\mathbf{o}_B$. This corresponds to a starting position of the three points

$${}^A\boldsymbol{p}_1^{\text{init}} = \begin{bmatrix} -0.10915 \\ -0.3976 \\ -0.253541 \end{bmatrix}, \qquad {}^A\boldsymbol{p}_2^{\text{init}} = \begin{bmatrix} -0.00915 \\ -0.2976 \\ -0.253541 \end{bmatrix}, \qquad {}^A\boldsymbol{p}_3^{\text{init}} = \begin{bmatrix} -0.10915 \\ -0.2976 \\ -0.153541 \end{bmatrix}, \tag{4.3}$$

with $\dot{\boldsymbol{p}}_i^{\text{init}} = 0_{6\times1}$ and $\ddot{\boldsymbol{p}}_i^{\text{init}} = 0_{6\times1}$ for every i. The three static (i.e. its time derivatives are zero) reference points $\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3$ are derived from a reference frame D represented by ${}^A\mathbf{H}_D$. To ensure that the set of reference points have the same relative distance as the end-effector-fixed points $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$, the reference points are chosen to lie on 0.1 m from the center of frame D. Note that this property is not required for (4.1) in order to execute the reach task. However, it does provide an insightful goal and a more predictable resulting motion. Given ${}^D\boldsymbol{d}_1, {}^D\boldsymbol{d}_2, {}^D\boldsymbol{d}_3$ as the three base vectors (length 0.1 m) of frame D expressed in frame D, the reference point $\boldsymbol{d}_i$ is expressed in frame A using the following homogeneous relation

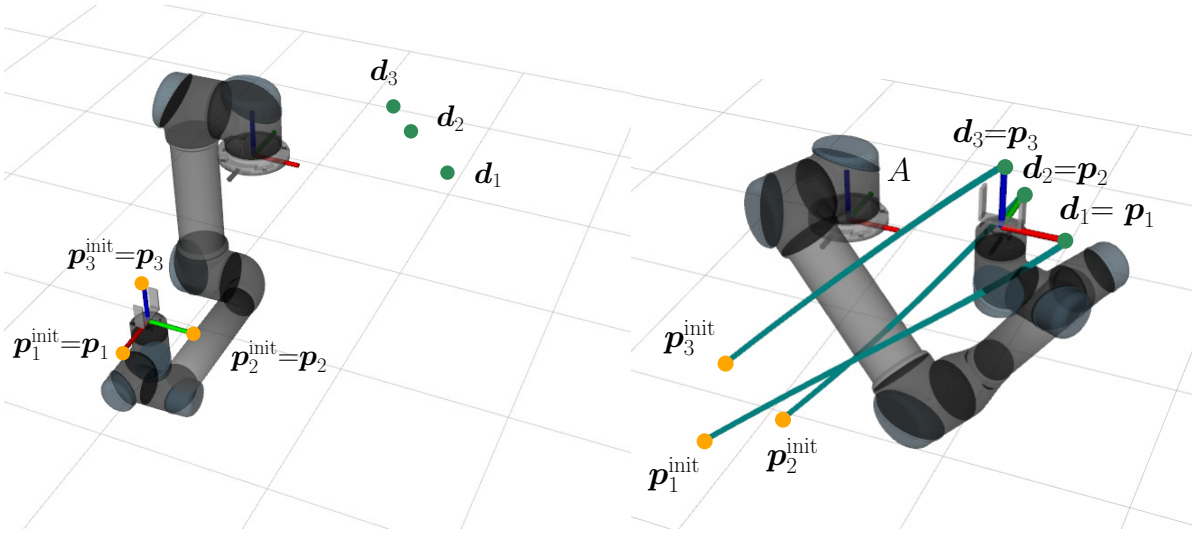
$${}^A\boldsymbol{d}_i = {}^A\mathbf{o}_D + {}^A\mathbf{R}_D\,{}^D\boldsymbol{d}_i, \tag{4.4}$$

resulting in

$${}^A\boldsymbol{d}_1 = \begin{bmatrix} 0.49085 \\ -0.2976 \\ 0.246459 \end{bmatrix}, \qquad {}^A\boldsymbol{d}_2 = \begin{bmatrix} 0.39085 \\ -0.1976 \\ 0.246459 \end{bmatrix}, \qquad {}^A\boldsymbol{d}_3 = \begin{bmatrix} 0.39085 \\ -0.2976 \\ 0.346459 \end{bmatrix}. \tag{4.5}$$

Figure 4.1 shows a visual representation of the end-effector-fixed and reference points.

Results. This section presents the results of the simulation of a pose reach task using the task-point approach. Figure 4.1a shows the initial configuration of the manipulator at the beginning of the simulation. Figure 4.1b shows the final configuration after completing the reach task. The resulting motion of the simulation is shown in Figure 4.1b as the cyan trails, starting from the initial points.



(a) (b)

Figure 4.1: *Figure (a) shows the initial configuration of the manipulator and initial states of the system. Figure (b) shows the final configuration of the manipulator. The cyan trails represent the resulting motion from the initial points* $\boldsymbol{p}_i^{init}$ *towards the reference points* $\boldsymbol{d}_i$*, executed on a six DoF manipulator.*

The cyan trails indicate a smooth motion of each end-effector-fixed point towards its corresponding reference point. However, it is hard to conclude anything substantial from the two static pictures other than the path taken by points. Further analysis is therefore done using the joint states in Figure 4.2 and the task space error in Figure 4.3 as a function of time.

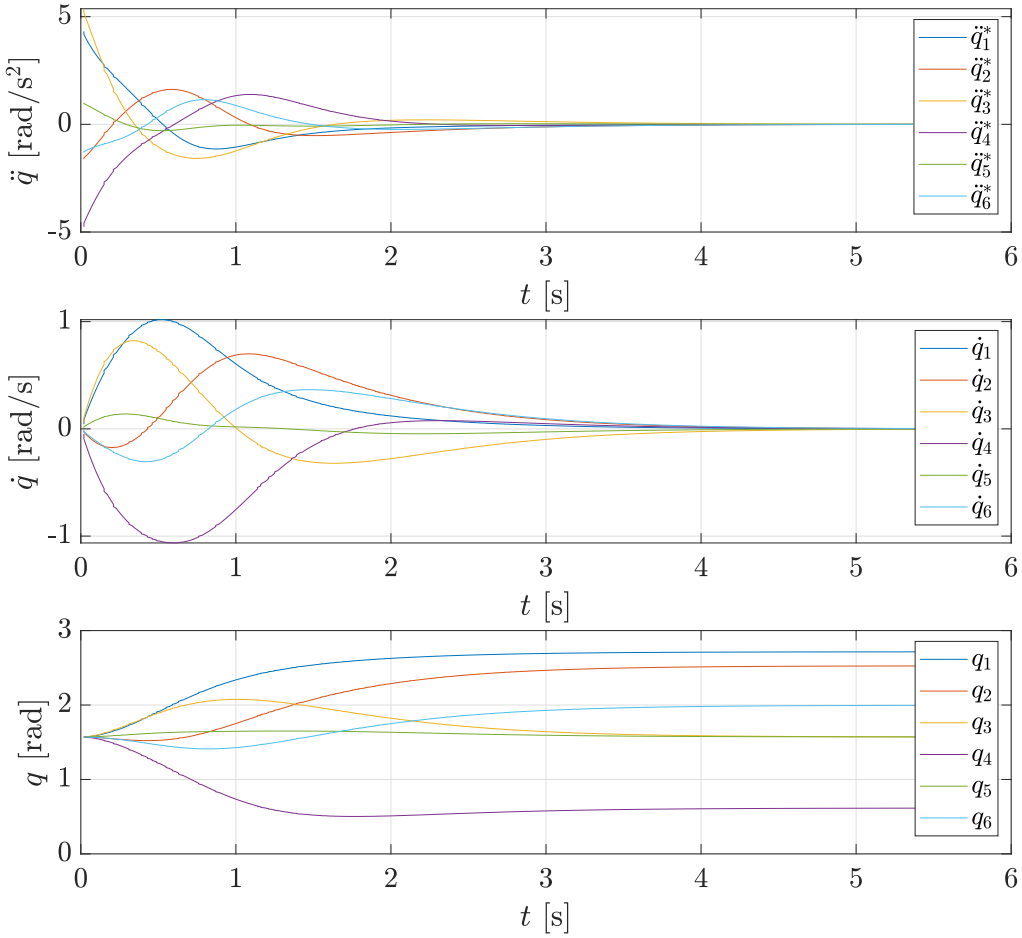


Figure 4.2: *The QP controller output $\ddot{\boldsymbol{q}}^*$, integrated to update the joint states $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$.*

The first plot shows the output of the QP controller $\ddot{\boldsymbol{q}}^*$ (i.e. the control effort). The figure shows that initially the QP controller has to do the most effort, which corresponds to the fact that the initial error is the largest, see Figure 4.3. The magnitude of the controller effort is mainly determined by the outputs of each of the PD controllers ${}^A\ddot{\boldsymbol{d}}_i^*$, since this is the acceleration the QP tries to achieve. Without any physical limits the added as constraints the QP problem and as long the three point tasks do not interfere with each other, the QP controller will generate an acceleration that matches the output of the PD controllers. The second and third plot are integrated from $\ddot{\boldsymbol{q}}^*$ in order to update the state of the robot $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$ respectively, which is required for the numerical computation of the Jacobian and its derivative.

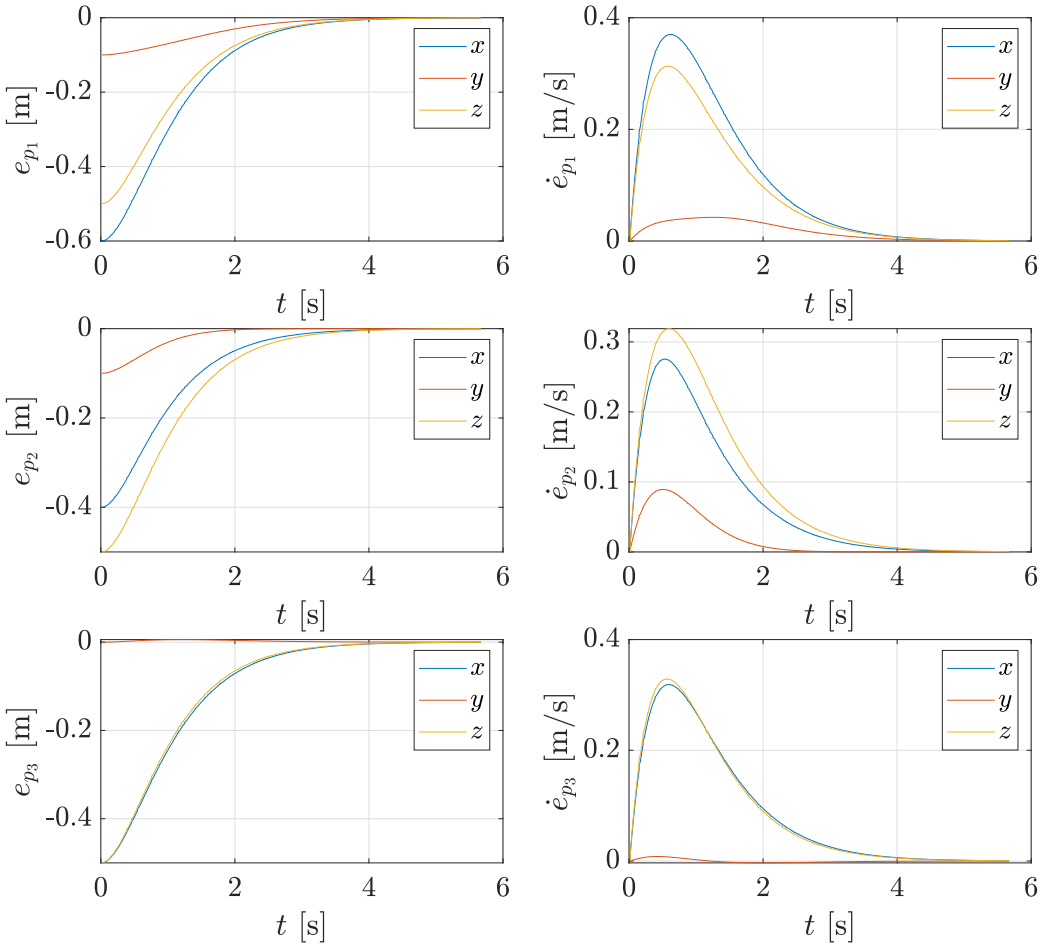


Figure 4.3: *Shows the position and velocity error ($\mathbf{e}_{p_i}$, $\dot{\mathbf{e}}_{p_i}$) for all three points as a function of time.*

This figure shows the position and velocity errors used in the PD controller of (4.1). The PD gains are chosen to generate a critically damped response, as discussed in Section 2.2.2, which is visible in the left column representing the position error response for each point. What the plots do not clearly depict is that the y component of the position error for the third point has a slight overshoot of 0.006 m and correspondingly the y component of the velocity error goes slightly below 0 m by 0.003 m. This behavior can be explained by the fact that the motion of each point separately has been designed to exhibit critically damped behavior, however, the QP controller employs a soft hierarchy and therefore cannot guarantee this behavior for each point, only a compromise defined as the cost function. This could be verified by analyzing the value of minimized cost of each of the points separately. If the value of the cost is not zero, there exists a slight deviation from the desired critically damped behavior. The figures also show that the task-point reach task is completed around the 4 s mark. This coincides with the steady state behavior of the joint states shown in Figure 4.2 and is most clearly visible in the joint velocity plot. The right column shows the velocity error and since the reference and initial velocity for each point is zero, the error also starts at zero. During the motion phase, until around 4 s, this

error cannot be zero or the end-effector is not able to move towards the reference points. This derivative error prevents large velocities of the end-effector and the convergence to a steady state solution.

4.3 Summary

The implementation of a task-point reach task was not a problem after the cumbersome process of understanding the ROS framework and TSID library. Singular configurations have been uncovered as an obstacle to numerically stable QP problems. However, under the right initial and goal conditions, the task-point reach task has been proven to successfully converge to its desired equilibrium state. The resulting motion of the end-effector and time evolution of its error both presented the expected behavior of a critically damped system.

Chapter 5

Conclusions & Recommendations

In this chapter, the conclusions that can be drawn from this research are discussed. A reflection on the main objective and its sub-objectives, as stated in Section 1.3, are made. Finally, recommendations regarding the research and future works are provided.

The main objective and the sub-objectives that are introduced in Section 1.3 are repeated here for convenience.

> *'Create a unified reach task formulation, compare this with a state-of-the-art reach task formulation in a QP control context and provide a proof of concept simulation of the unified reach task formulation.'*

Which has been divided into the five sub-objectives:

1. Understand the state-of-the-art reach task formulation and structure the obtained knowledge.
2. Create a unified reach task formulation.
3. Compare the state-of-the-art with the unified approach.
4. Software implementation of the unified reach task formulation.
5. Provide a proof of concept simulation of the unified reach task formulation.

5.1 Conclusions

The literature review in Section 1.2, has provided the state-of-the art in reach task formulations as a QP problem using the homogeneous transformation matrix to represent the pose of a part of the robot. In Section 2.2, QP control has been investigated and found that a QP controller on acceleration level results in smoother motions of the robot compared to a QP controller on velocity level. Additionally, a QP controller on acceleration level allows adding the dynamics of the robot if desired, because the dynamics are linear in the joint accelerations. In Section 2.2.3, PD control on SE(3) has been investigated and used to derive the cost function for the SE(3) reach task formulation in Chapter 3. The successful formulation of a SE(3) reach task as a QP problem implies that the first sub-objective has been reached sufficiently. More knowledge can still be obtained about other (redundant) reach task representations, such as the $\mathrm{SO}(3) \times \mathbb{R}^3$ and the reduced orientation ($\mathbb{S}^2$) representation discussed in Section 1.2. This knowledge can

then be used to create reach task formulations for these representations in order to provide a more elaborate comparison of the task-point approach with the existing literature.

In Chapter 3, the task-point reach task formulation has been developed using a set of points to represent the current and reference pose of a part of the robot. A single optimal control problem is created by adding the weighted norm of the cost function for each point in a QP controller. The weighted summation creates a soft priority hierarchy. The task-point reach task approach has proven to be quite flexible in specifying (redundant) reach tasks. The two main parameters in the specification are the amount of points and the location of these points, which has resulted in the specification of several (redundant) reach tasks: pose, position, position + direction, orientation and direction. This result successfully completes the second sub-objective to the extend of all basic reach task primitives. Furthermore, basic combinations of the reach task primitives have been made, however, no research was done into more complex combinations and its connection to practical applications. The task-point approach provides a unified and intuitive approach to the specification of (redundant) reach tasks and adds to existing literature in this regard. Especially, since existing literature has mainly focused on developing specific approaches to specific reach tasks. The task-point approach lends its self for further exploration as explained in the next section.

Chapter 3 also provides a detailed theoretical comparison between the SE(3) reach task and task-point reach task. The comparison mainly contributes to this research by investigating the effects of the parameters of the reach task formulations on the resulting motion. Not surprisingly, the resulting motion is mainly influenced by the magnitude of PD gains in combination with the error definition of both approaches. The SE(3) approach has a less intuitive error definition making the prediction of the resulting motion and therefore the tuning of the PD gains harder. The task-point approach provides the an intuitive error definition and tuning due to its simplicity and direct relation with the PD gains. The task-point approach removes the issue of choosing a weight between the units of acceleration and angular acceleration of the SE(3) approach. However, adds the parameter of deciding the relative placement of each point, which in turn also effects the relative importance between the position and orientation error. Simulations would need to be performed to analyze the tunability of the two approaches in this regard. Regarding the third sub-objective, both approaches have been compared on a basic theoretical level and add to the theoretical foundation of the task-point reach task. However, the foundation lacks stability proofs when considering multiple points and simulations to further the discussion on tunability of both approaches.

In Section 2.3, the software architecture used for the implementation of the SE(3) and task-point reach tasks has been provided in the form of a ROS framework using the TSID and Pinocchio libraries to generate and solve the QP problem. In Chapter 4, the task-point reach task formulation has been successfully implemented in this framework. The SE(3) reach task is already a part of the TSID library, however, the documentation on how exactly is missing. A proof of concept simulation of the task-point reach task approach in the form of a pose reach task has been performed. Overall, the results show the expected behavior of a critically damped system for each of the three points, however, the y-component of one point has an overshoot of 0.006 m. The small overshoot might be a result of conflicting interactions between the points during the motion. This could be verified by analyzing the value of minimized cost of each of the points separately. If the value of the cost is not zero, there exists a slight deviation from the

desired critically damped behavior. A proof of concept has only been shown for a pose reach task, however, the other basic (redundant) reach tasks still require a proof of concept simulation. Also, the pre-processing step (3.4), required for the specification of orientation reach tasks, has not been implemented yet. Therefore, regarding the sub-objectives four and five, only a first step in the proof of concept of the task-point reach task has been made and leaves room for future research.

5.2 Recommendations

As concluded in the previous section, the main research objective of this thesis has been achieved on a theoretical level and the first steps towards a proof of concept have been made. Several other topics on future research can also be derived from the current findings in this research. Therefore, a list of recommendations on future work is presented.

Task-Point Reach Tasks. The task-point reach task formulation has only just been conceived and only the most basic reach tasks have been formulated. It is therefore interesting to draw inspiration from practical applications in the field of robotics for more complex reach tasks or to use combinations of these basic reach tasks in order to form more complicated ones.

Physical Robot. The current discussion on reach tasks has been mainly focused on the task formulation and the kinematics of the problem. However, reach tasks eventually need be executed on a real physical robot, which brings several new issues to the reach task problem. Firstly, in the case that the robot's joints are torque controlled, the dynamics of robot can be used to transform the output of the QP controller (joint acceleration) to joint torques by adding the dynamics as an equality constraint. Secondly, the dynamics of the robot can also be used to relate the joint positions, velocities and accelerations to joint torques bounds that represent the actuator limits of the motors in the joints. These can be implemented as inequality constraints to the QP control formulation. Thirdly, most joints have certain upper and lower bounds to their range of motion. These can again be added as inequality constraints, however, their relation to joint accelerations is not trivial. Several strategies for joint position and velocity bounds are discussed in [32]. Fourthly, currently the QP controller does not consider any self-collisions or collisions with its environment. In this regard, often an environment representation using depth information together with potential field methods are employed to avoid self-collisions and collisions with its environment. However, potential field methods do suffer from local minima if not carefully designed. Besides the main task, potentially conflicting tasks such as collision avoidance tasks can be added to the QP problem as an additional task in the form of a cost function. Another option is to switch to a collision avoidance task for the duration of a potential collision and then switching back to the main reach task. Finally, as mentioned in Chapter 4, singular configurations can be a cause for unpredictable and unstable behavior. It is therefore useful to avoid these singularities. A possible solution could be to quantify a "distance" to singularities and create an avoidance strategy, perhaps similar to a joint bound. Or plan "smart" global paths that avoid these singularities.

Numerical Simulations. Numerical simulations are an important tool in the validation of models and control strategies. In the case of the task-point reach task a proof of concept was the first step towards the validation of the proposed control strategy. Other interesting simulations include, testing robustness against perturbations, inconsistent task-point references

(i.e. the relative distances among a set of reference points and a set of link-fixed points are not equal) and the investigation into task-point singularities, as discussed in Section 3.5. Once the task-point reach has been fully implemented and validated, it should be compared to other state-of-the-art solutions to compare performance metrics such as the convergence rate and how predictable the resulting motion is based on the tuning of its parameters.

Experiments. When the task-point reach task formulation has been successfully validated, physical experiments can be performed. Since the formulation currently only considers the kinematics of the robot, good care in the design of the reach task and controller gains should be employed to ensure safe operations of the robot. An interesting task is the task of reaching for a cup on a table to highlight the effects of redundancy of the reach task on the robot in a physical world.

Bibliography

[1] Tech united. `http://www.techunited.nl/`, 2019.

[2] Bot and dolly. `http://www.botndolly.com/`, 2019.

[3] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2005.

[4] Robocup. `www.robocup.org`.

[5] J. Vaillant, A. Kheddar, H. Audren, F. Keith, S. Brossette, K. Kaneko, M. Morisawa, E. Yoshida, and F. Kanehiro. Vertical ladder climbing by the hrp-2 humanoid robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 671–676, Nov 2014.

[6] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson. Optimization based full body control for the atlas robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 120–127, Nov 2014.

[7] N. A. Chaturvedi, A. K. Sanyal, and N. H. McClamroch. Rigid-body attitude control. *IEEE Control Systems Magazine*, 31(3):30–51, June 2011.

[8] R. Campa and H. de la Torre. Pose control of robot manipulators using different orientation representations: A comparative review. In *2009 American Control Conference*, pages 2855–2860, June 2009.

[9] Sanjay P. Bhat and Dennis S. Bernstein. A topological obstruction to continuous global stabilization of rotational motion and the unwinding phenomenon. *Systems & Control Letters*, 39(1):63 – 70, 2000.

[10] F. Bullo, R.M. Murray, and A. Sarti. Control on the sphere and reduced attitude stabilization. *IFAC Proceedings Volumes*, 28(14):495 – 501, 1995. 3rd IFAC Symposium on Nonlinear Control Systems Design 1995, Tahoe City, CA, USA, 25-28 June 1995.

[11] F. Bullo and R. M. Murray. Proportional derivative (pd) control on the euclidean group. *Proceedings of the 3rd European Control Conference*, page 1091–1097, 1995.

[12] Andrea Del Prete. *Control of Contact Forces using Whole-Body Force and Tactile Sensors: Theory and Implementation on the iCub Humanoid Robot*. PhD thesis, 04 2013.

[13] A. Rocchi, E. M. Hoffman, D. G. Caldwell, and N. G. Tsagarakis. Opensot: A whole-body control library for the compliant humanoid robot coman. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6248–6253, May 2015.

[14] O. Kanoun, F. Lamiraux, and P. B. Wieber. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27(4):785–792, Aug 2011.

[15] Alexander Dietrich, Christian Ott, and Alin Albu-Schäffer. An overview of null space projections for redundant, torque-controlled robots. *The International Journal of Robotics Research*, 34(11):1385–1400, 2015.

[16] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. In *2009 International Conference on Advanced Robotics*, pages 1–6, June 2009.

[17] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, 2014.

[18] C. Samson and B. Espiau. Application of the task-function approach to sensor-based control of robot manipulators. *IFAC Proceedings Volumes*, 23(8, Part 5):269 – 274, 1990. 11th IFAC World Congress on Automatic Control, Tallinn, 1990 - Volume 5, Tallinn, Finland.

[19] A. Saccon S. Traversaro. Multibody dynamics notation (revision 2). Available online at `tue.research.nl`, 2019.

[20] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[21] C. T. J. Beumer. Impact aware robot manipulation via task-based reference spreading. 2019.

[22] Ros control. `http://ros.org`.

[23] Ros: Urdf explained. `http://wiki.ros.org/urdf/XML/joint`.

[24] LAAS CNRS. Task-space inverse dynamics (tsid) software library. `https://github.com/stack-of-tasks/tsid`.

[25] INRIA and LAAS CNRS. Pinocchio. `https://github.com/stack-of-tasks/pinocchio`.

[26] LAAS CNRS. Tsid hqp eiquadprog solver. `https://github.com/stack-of-tasks/tsid/blob/master/src/solvers/solver-HQP-eiquadprog.cpp`.

[27] Tsid uquadprog++. `https://github.com/stack-of-tasks/tsid/blob/master/include/tsid/solvers/eiquadprog_2011.hpp`.

[28] L. Žlajpah. On orientation control of functional redundant robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2475–2482, May 2017.

[29] James O. Coplien. Curiously recurring template patterns. *C++ Rep.*, 7(2):24–27, February 1995.

[30] Peter Canning, William Cook, Walter Hill, and Walter Olthoff. Abstract f-bounded polymorphism for object-oriented programming, 1989.

[31] Jim Beveridge. Understanding atl's atypical design approach. `https://web.archive.org/web/20060315072824/http://www.apostate.com/programming/atlupsidedown.html`.

[32] A. D. Prete. Joint position and velocity bounds in discrete-time acceleration/torque control of robot manipulators. *IEEE Robotics and Automation Letters*, 3(1):281–288, Jan 2018.

Technische Universiteit
Eindhoven
University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct [i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

18-06-2019

Name

Rik van der Struijk

ID-number

0739222

Signature

Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU