

Documentation of RoSta Task Specification Case Study

Tomas Olsson

September 21, 2007

Abstract

The primary purpose of this document is to describe the current status of the RoSta case study on constraint-based robotics task specification, utilizing the modelling framework described in [3]. The secondary purpose is to provide an outline for future work, extensions, and improvements.

1. Test Case Setup

The test case is based on a mobile robot arm, which is to interact with a workpiece using guidance from a force/torque sensor, a laser range sensor, and a camera. Since the focus is on task specification rather than on control and estimation problems, the test case has been chosen such that all controlled outputs are directly measurable.

The simulated test cell, shown in Fig. 1, consists of a 6-DoF robot equipped with a tool with an integrated camera and a tripod pressure foot, each foot sliding freely on the surface. A planar workpiece is placed on a table in the work-cell. On the planar workpiece two accurately pre-drilled holes serve as reference features for the camera-based positioning system. The force/torque sensor measures the total forces and torques on the pressure foot, facilitating control of the alignment torques as well as the axial force.

A Comment on Notation

In the following, the notation of the type $T_{f_i}^{o_i}$ is used to denote the rigid transformation from a coordinate system (frame) o_i to the frame f_i . Although the representation for the rigid transformation could in principle be chosen in any of a number of different ways (quaternions, translation vectors, angle-axis, Euler angles...), in the following is assumed to be a 4×4 rigid transformation matrix. The reason is that the relation to the corresponding coordinate descriptions of the transformation is straightforward.

Feature and Object Frames

Four features are defined, one for the tripod contact (feature a), one for each of the two reference features (holes) in the workpiece (features b and c), and one for the laser range sensor (feature d). All features share a common definition of the object frames o_1 and o_2 , which are rigidly attached to the workpiece and the flange of the robot, respectively, as shown in Fig. 2.

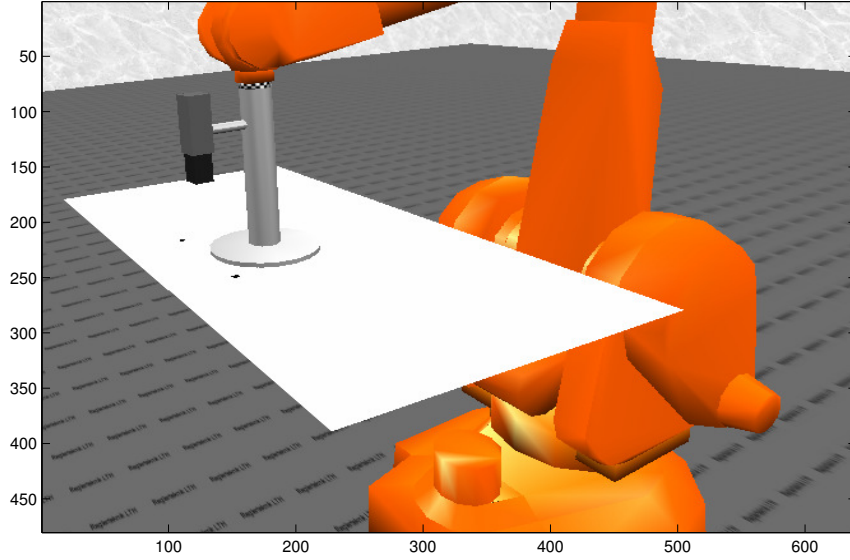


Figure 1 Visualization of the simulated test cell.

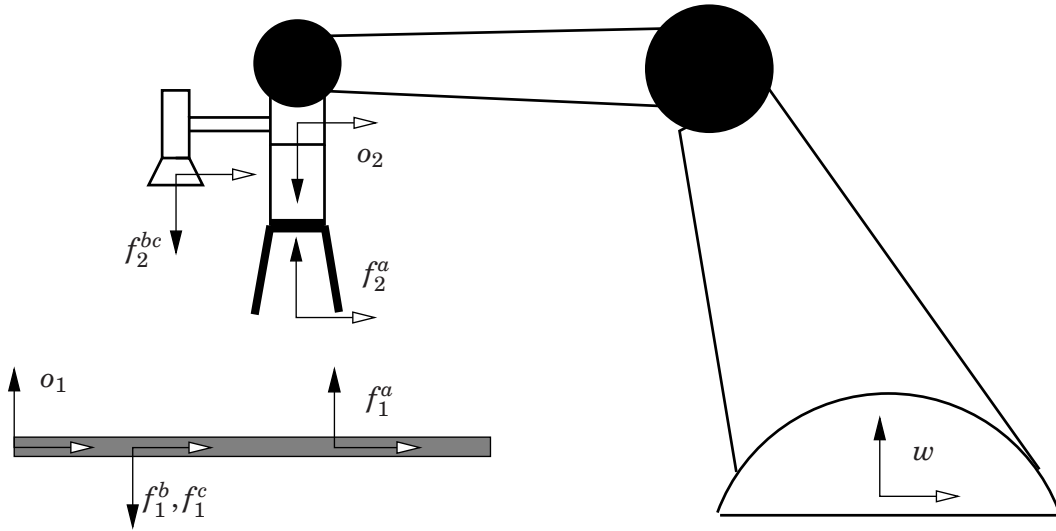


Figure 2 Feature and object frames in test setup, with the direction of z-axes (solid arrows) and x-axes of each frame indicated.

Feature a. Feature frame f_1^a is attached to the workpiece at the desired alignment position for the TCP, defined as the *center of the tripod*, and with its orientation the same as for o_1 . Feature frame f_2^a is attached to the drilling tool, with its origin at the TCP point and oriented such that *in the desired aligned position* the orientation is the same as for f_1^a .

Feature b and c. Feature frame f_2^{bc} are attached to the camera, with origin at the center of projection and the z-axis forward. Feature frames f_1^b and f_1^c are attached to the workpiece at the drilled reference holes, oriented such that *in the desired aligned position* the orientations are the same as for f_2^{bc} .

Feature d. For simplicity, the sensor is assumed to have been accurately mounted with its axis along the z-axis of the flange frame o_2 , which means that the sensor location can (with a simple transformation of the range

measurement) be modeled to coincide with object frame o_2 . Feature frames f_1^d and f_2^d have their origins at the intersection point between the laser beam and the workpiece surface, with the same orientations as o_1 and o_2 , respectively.

Task Coordinates

For the first three features, all degrees of freedom are placed between frames f_1 and f_2 . Therefore, for each feature the task coordinates are chosen as a vector χ_{fII} of translations and roll-pitch-yaw (rpy) angles.

Feature a. The task coordinates are defined as the rpy/translation representation of the transformation from frame f_2^a to frame f_1^a , denoted by

$$\chi_{fII}^a = (x_a \ y_a \ z_a \ \phi_a \ \theta_a \ \psi_a)^T. \quad (1)$$

Feature b and c. The task coordinates are defined as the rpy/translation representations of the transformations from frame f_1^b to frame f_2^{bc} and from f_1^c to f_2^{bc} , denoted by

$$\chi_{fII}^b = (x_b \ y_b \ z_b \ \phi_b \ \theta_b \ \psi_b)^T \quad (2)$$

and

$$\chi_{fII}^c = (x_c \ y_c \ z_c \ \phi_c \ \theta_c \ \psi_c)^T. \quad (3)$$

Feature d. The task coordinates for feature d are divided into three parts. χ_{fI}^d are defined as the x and y coordinates of the laser-workpiece intersection point measured in object frame o_1 . χ_{fII}^d are the roll-pitch-yaw angles of the transformation f_2^d to f_1^d . χ_{fIII}^d is the z-coordinate z_d of the intersection point as measured in object frame o_2 . Together, these parameters fully define the six degrees of freedom of the relative sensor-workpiece transformation.

Uncertainty coordinates

Uncertainty coordinates are introduced to describe errors in the workpiece positioning, the robot kinematics, the tool TCP point, and the camera mount. The following uncertainties are defined, all described as 6-DoF rpy/translation representations of the corresponding transformations:

- χ_{uI} represents the transformation from the modeled workpiece frame o_1' to the true frame o_1 .
- χ_{uIII}^a represents the error transformation, $T_e^a \stackrel{\text{def.}}{=} T_{f_2^{o_2}}^{o_2} T_{o_2'}^{f_2^a}$, for the TCP frame with respect to the flange frame.
- χ_{uIII}^{bc} represents the error transformation, $T_e^{bc} \stackrel{\text{def.}}{=} T_{f_2^{o_2}}^{o_2} T_{o_2'}^{f_2^{bc}}$, for the camera frame with respect to the flange frame.
- χ_{uIV} represents the transformation from the modeled flange frame o_2' to the true frame o_2 .

Note that the definition for χ_{uIII} differs from the one suggested in [3]. The new definition is more natural in this application, since it separates the error in the tool calibration from the kinematics errors parametrized by χ_{uIV} .

Position Loop Constraints

There are four loop constraints, expressed as constraint functions

$$l(q, \chi_f, \chi_u) = 0 \quad (4)$$

of task coordinates, uncertainties and robot configurations q . Each of the first three loop constraint functions is a relation in the form

$$l_i(q, \chi_f, \chi_u) = \text{rpy} \left(T_w^{o'_1} T_{o'_1}^{o'_1}(\chi_{uI})^{-1} T_{o'_1}^{f_1} T_{f_1}^{f_2}(\chi_{fII}) T_e(\chi_{uIII}) T_{f_2}^{o'_2} T_{o'_2}^{o'_2}(\chi_{uIV}) T_{o'_2}^w(q) \right), \quad (5)$$

expressed by computing the rpy/translation representation of the (ideally closed) chain of transformations. As no uncertainty coordinates need to be introduced for the laser range sensor position, the loop constraint for feature d takes the simplified form

$$T_w^{o'_1} T_{o'_1}^{o'_1}(\chi_{uI})^{-1} T_{o'_1}^{f_1^d}(\chi_{fI}^d) T_{f_1^d}^{f_2^d}(\chi_{fII}^d) T_{f_2^d}^{o'_2}(\chi_{fIII}^d) T_{o'_2}^{o'_2}(\chi_{uIV}) T_{o'_2}^w(q) = I. \quad (6)$$

Outputs and Measurements

In the example, the outputs y and measurements z are identical. The outputs are the axial z-force and the aligning x and y-torques acting on the TCP point, as measured in f_2^a -coordinates, as well as the projected image coordinates of the reference holes in the camera. The forces are assumed to be related to the deformations by a linear spring characteristic, giving the force measurements

$$z_1 = k_z z_a \quad (7)$$

$$z_2 = k_\phi \phi_a, \quad z_3 = k_\theta \theta_a. \quad (8)$$

The camera is assumed to be internally calibrated and normalized, giving the measurements

$$z_4 = x_b/z_b, \quad z_5 = y_b/z_b \quad (9)$$

$$z_6 = x_c/z_c, \quad z_7 = y_c/z_c. \quad (10)$$

The measurement from the measured laser range is

$$z_8 = z_d, \quad (11)$$

the z-coordinate of the intersection point as measured in object frame o_2 . The references are assumed to be constant, where the force references are chosen as

$$y_{1d} = F_{zd} \quad (12)$$

$$y_{2d} = y_{3d} = 0, \quad (13)$$

while the values for the image references $y_{4d}, y_{5d}, y_{6d}, y_{7d}$ are assumed to have been acquired by teaching. This means that it can be assumed possible to reach the specified references, despite the geometric uncertainty.

Mobile Platform

In order to test the ability to model mobile and non-holonomic robots, the robot arm is put on a cart, with two differentially driven wheels. Somewhat simplistically it is assumed that the orientation and position of the cart can be measured exactly and directly, for instance using dead-reckoning or some external positioning system. The setup, with object and feature frames indicated, can be seen in Fig. 3.

The configuration of the arm q is augmented into the new robot coordinates χ_q with the parameters x_r, y_r and ψ_r , describing the planar translation and z-rotation from the robot base frame to the robot-fixed frame r . Both the

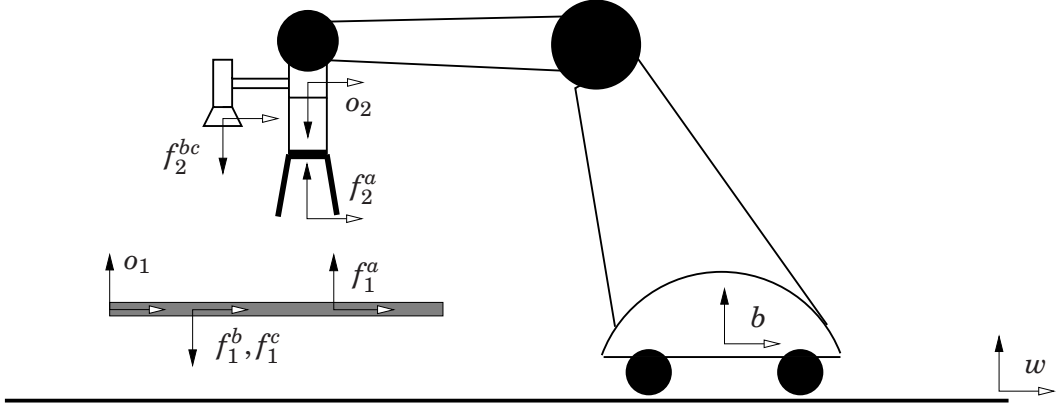


Figure 3 Feature and object frames in test setup, with the direction of z-axes (solid arrows) and x-axes of each frame indicated.

world frame w and frame r are oriented with their z-axis vertically. The new measurements are functions of the robot configuration

$$z_9 = x_r, \quad z_{10} = y_r, \quad z_{11} = \psi_r. \quad (14)$$

The time derivatives of the new configuration parameters (and measurements) are related to the actuated wheel velocities through the linear relationship

$$\begin{pmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\psi}_r \end{pmatrix} = \begin{pmatrix} -\frac{R}{2} - y_r \frac{R}{B} & -\frac{R}{2} + y_r \frac{R}{B} \\ x_r \frac{R}{B} & -x_r \frac{R}{B} \\ \frac{R}{B} & -\frac{R}{B} \end{pmatrix} \begin{pmatrix} \dot{q}_l \\ \dot{q}_r \end{pmatrix}, \quad (15)$$

which is used in the control by modifying the Jacobians as described in Section 5.2 in [3].

Simulations

A number of simulations have been performed in Matlab using a robot with ideal velocity dynamics, and visualized using computer graphics tools. Direct control using the measurable outputs was used, as in [3]. Small, constant, unmodeled disturbances were added in the uncertainty coordinates, in order to simulate imperfect calibration. The resulting convergence of the system outputs to the desired references can be seen in Fig. 4. The simulation results for the cart positioning control can be seen in Fig. 5, for a simple proportional control which is still able to drive the cart to a configuration close to its desired position.

2. Representation and Implementation

This section describes an initial attempt to standardize the representation of robotics tasks, such as the setup described above. Standardization is all about getting to an agreement about *how to represent* “information,” that is, the meaning of, both, *data* and of the *algorithms* that process these data. For example, the displacement of a robot with respect to a world reference (“data”), and the composition of two such displacements (“algorithm”). A good standard in a particular domain then satisfies the following information representation requirements:

- it covers *all objects* that live in its domain.

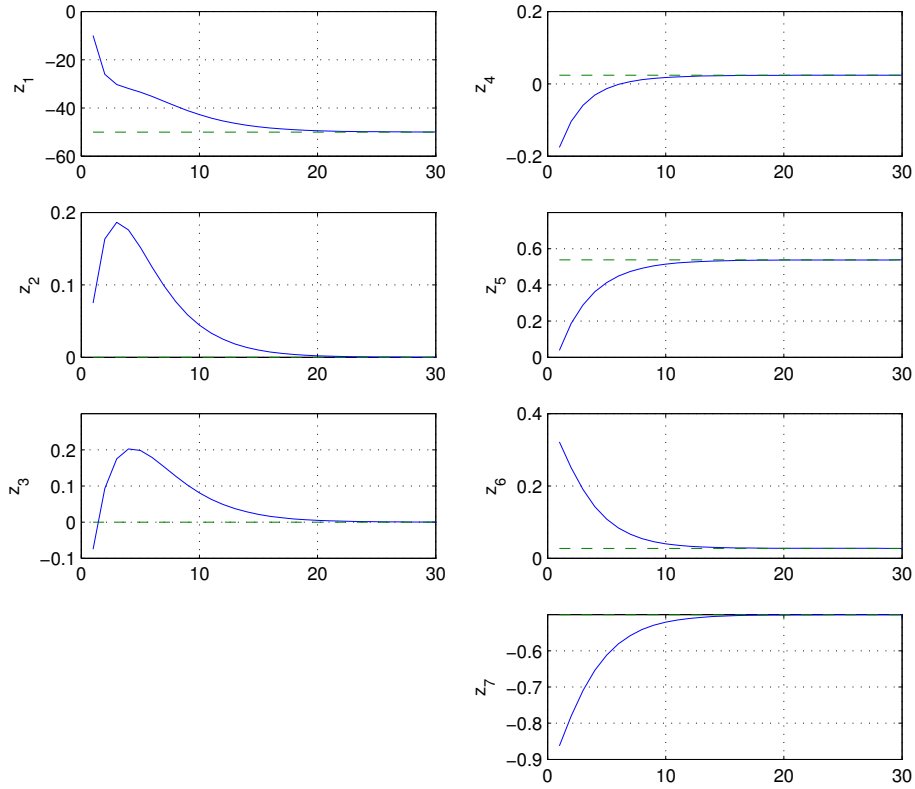


Figure 4 Convergence of the true outputs z (*solid lines*) to constant references y_d (*dashed lines*).

- it creates *no ambiguities* in how the represented information is to be interpreted, by both humans (experts in the domain) and computers.
- it *allows efficient implementations* for all “use cases” in its domain.

In most cases, no single information representation can achieve all possible specific requirements for all applications. Hence, standardization makers should adapt to this reality, and prepare a “family” of complementary standards that share as many as possible of the four generic levels discussed below.

1. *Ontology representation*, defining *in words*, independently of any mathematical representation, the terminology and the meaning (“semantics”) of the relevant objects in the scope of the standard, and of the natural operations on those objects.
2. *Mathematical representation*, providing data structures (“coordinate” representations) for the above-mentioned objects, as well as the API (Application Programming Interface) for the natural operations on the physical objects.
3. *Computer representation*, defining how coordinates are represented in computer-readable form.
4. *Native hardware representation*, defining the mapping from computer-readable variables into bit representations in the hardware.

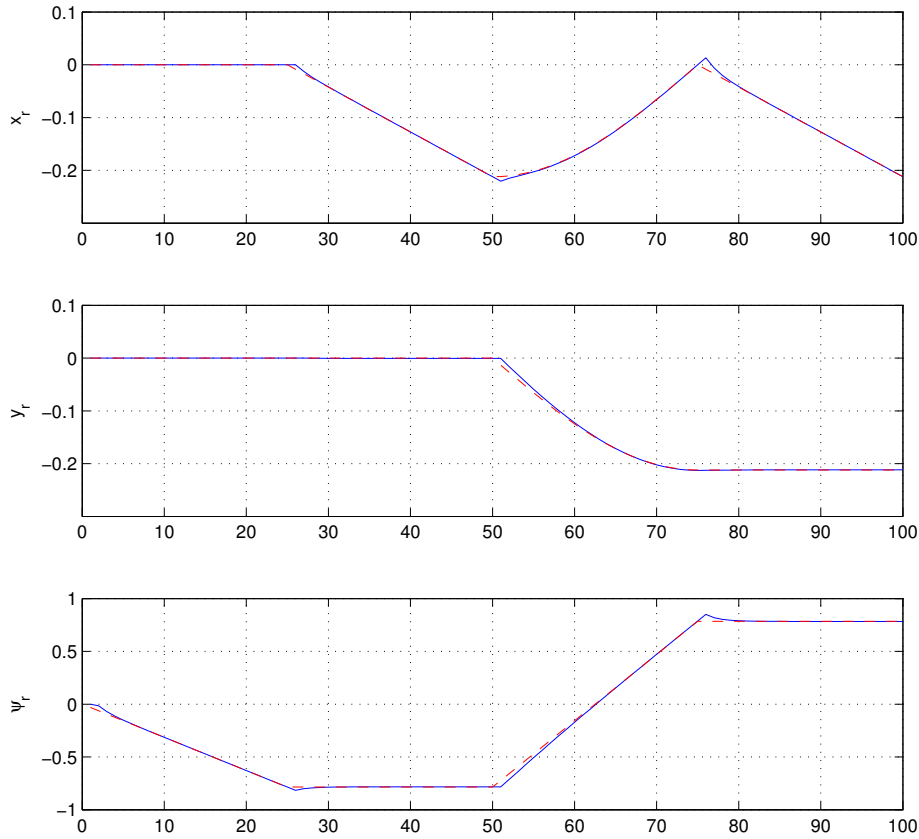


Figure 5 True (*solid lines*) and references (*dashed lines*) for the cart configuration parameters x_r , y_r and ψ_r .

In the following a description of the mathematical and computer representations for the task specification case is given. Although not all of these levels have been covered in this example, some pointers to what would be required are given in the following sections.

Mathematical Representation

The purpose of the mathematical description is to provide sufficient information to completely specify a task, information which could be used for both analysis and for automatic code generation (e.g. using Maple) of the functions required in the implementation of a controller. Although Maple code has been used to exemplify below, all mathematical expressions could (and should) equivalently be represented using a portable and platform-independent open format such as MathML.

A full mathematical task specification should contain at least the following components, although not all parts have been included in the example:

Support functions and data structures. The mathematical expressions for basic operations such as creation, composition, and inversion of rigid transformations need to be clearly defined. In the example, we assume the existence of a separate set of Basic Robotics Standards (BRoS) for rigid objects, transformations, and kinematic chains, as outlined in [4]. Although not part of the task specification itself, in the example some

basic operations for rotation matrices have been defined mathematically.

Coordinates. The task coordinates χ_f , robot operational space (or joint) coordinates χ_q , and uncertainty coordinates χ_u , should be defined. On the mathematical level, this is done by specifying vectors of symbolic variables, giving the range of each coordinate vector:

```
> Xq := array(1 .. 9);
> Xu := array(1 .. 30);
> Xf := array(1 .. 24);
```

Constant parameters and functions. The mathematical description contains a number of parameters that describe *constant* (i.e. coordinate-independent) transformations and other functions. In some cases, it would be beneficial to make some of these parameters tunable on the control level. Although this has currently not been implemented, the symbolic form of the mathematical expressions make it relatively straightforward to specify and include also tunable parameters. Further, there are functions that describe time-varying constraints, geometrical surfaces, and robot kinematics. These functions should be possible to express either on symbolic analytical form, implicitly (to be resolved on the implementation level, for instance in the case of non-analytical PKM kinematics), or possibly as structures and algorithms from other domains (for instance in the case of spline and 3D shape descriptions of geometry).

Dynamic model. At least some parts of the dynamics that govern the motion of the robot and other dynamical components should be considered a part of the task model, and be included in ODE or DAE form. Similarly, dynamical models for the geometrical uncertainty χ_u should also be specified. Standardized formats for the dynamics have not been developed for the example.

Objects and coordinate-dependent transformations. All task-relevant objects, such as robots, tools, and workpieces, should be explicitly defined, based on the suggested data structures in the BRoS standard [4]. For each object, its base frame relative to the world is specified, as well as any number of (base-relative) handle frames used to connect the object to other components in the workspace. Each frame or transformation is allowed to depend on the coordinates in χ_f , χ_q , and χ_u . For example, the following expressions give the code for the work object, its object frame o_1 , and the feature frames $f_1^a-f_1^d$:

```
> To1T0w := multiply(rpy2Rt(0,0,0,-1,0,1),invRt(
    rpy2Rt(Xu[1],Xu[2],Xu[3],Xu[4],Xu[5],Xu[6])));
> Tf1aT0o1 := rpy2Rt(0, 0, 0, 0.2, 0, 0);
> Tf1bT0o1 := rpy2Rt(0, 0, Pi, 0.1, -0.1, 0.1e-2);
> Tf1cT0o1 := rpy2Rt(0, 0, Pi, 0.1, 0.1, 0.1e-2);
> Tf1dT0o1 := rpy2Rt(0, 0, 0, Xf[19], Xf[20], 0);
> To2T0o2 := rpy2Rt(0, 0, 0, 0, 0, 0);
> object := [eval(To1T0w), [eval(Tf1aT0o1), eval(Tf1bT0o1),
    eval(Tf1cT0o1), eval(Tf1dT0o1)]];
```

Generalized joints. Each object or component in the workspace is connected to other components through generalized joints, describing the transformation between two handles on different components. The following code defines the transformations linking feature frames $f_1^a-f_1^d$ to feature frames $f_2^a-f_2^d$, as well as a “dummy” joint linking the tool to the robot wrist:

```

> Tf2aT0f1a := rpy2Rt(Xf[1], Xf[2], Xf[3],
                    Xf[4], Xf[5], Xf[6]);
> Tf2bT0f1b := invRt(rpy2Rt(Xf[7], Xf[8], Xf[9],
                    Xf[10], Xf[11], Xf[12]));
> Tf2cT0f1c := invRt(rpy2Rt(Xf[13], Xf[14], Xf[15],
                    Xf[16], Xf[17], Xf[18]));
> Tf2dT0f1d := rpy2Rt(Xf[21], Xf[22], Xf[23], 0, 0, 0);
> joint1 := [eval(Tf2aT0f1a)];
> joint2 := [eval(Tf2bT0f1b)];
> joint3 := [eval(Tf2cT0f1c)];
> joint4 := [eval(Tf2dT0f1d)];
> joint5 := [eval(To2T0o2)];

```

Kinematic chains and position loop constraints. Using the definition of objects and joints one kinematic chain for each feature is defined, linking task, robot, and uncertainty coordinates. Together with a criterion function, indicating the difference between the transformation from the identity (indicating a closed chain), this specifies the constraint functions $l(\mathcal{X}_q, \mathcal{X}_f, \mathcal{X}_u)$ in [3]. These functions are used in the controller and estimator, in order to obtain *consistent* sets of coordinate values from the the over-parametrized coordinate descriptions defined above. The following defines one of the four position loop constraints, the loop for feature a , as a sequence of handles and joints connecting the handles:

```

> kinchain_a := [[[robot, 1], joint5, [tool, 1],
                [tool, 2], joint1, [object, 1]]];

```

Controlled outputs. The controlled outputs should be specified as symbolic functions of task coordinates and robot operational space coordinates, in a straightforward way.

```

> h := [kz*Xf[6], krx*Xf[3], kry*Xf[2], Xf[10]/Xf[12],
        Xf[11]/Xf[12], Xf[16]/Xf[18], Xf[17]/Xf[18],
        Xf[24], Xq[7], Xq[8], Xq[9]]);

```

Measurements. Similarly, the available measurements are specified as functions of task coordinates and operational space coordinates.

Reference constraints. In order to allow also for references from external sources, the exact values for reference signals should not be required in the mathematical task specification. Instead, and more generally, it should be possible to specify equality or inequality constraints for the references and their corresponding velocities/accelerations. The constraints should be used in a reference generator module on the control level.

Computer Representation

Similarly to the mathematical representation being encoded using MathML, an open system for describing the computer representation of the data should be used. The language and communication protocol LabComm, developed for robotics applications at Lund University, could be used to this purpose. In LabComm, data definitions are written in a C-like syntax, used to generate a protocol library which can be used to automatically generate C or Java code to be integrated into the application. Among the advantages of having the data definition in Labcomm are

- The LabComm data definitions provide a clear, platform independent way to specify the data on the computer representation level, as well as functionality for consistency and type checking.

- LabComm data definitions (or a subset thereof) can be converted automatically into a number of different formats such as MathML, or read into tools such as Maple for use as input/output type definitions in a system for automatic code generation.
- Given the protocol/data description, the LabComm compiler can be used to automatically generate C or Java code for the communication protocol, greatly simplifying interfacing and integration with other applications.

In the example, the computer level representation of the algorithms defined on the mathematical level, are automatically generated through the Maple CodeGeneration package. Currently, some extra manual operations are necessary in order to create the parameter interfaces, although all task specification-dependent code is generated automatically. The API data definitions could be obtained for instance from a LabComm definition file, although this is currently not implemented. The generated C code is then compiled into Matlab mex-functions, to be called from the Matlab controller.

Implementation details

The mathematical representation of the task can be found in the Maple workspace file `RoSta_TaskSpec.mw`. This file contains the following components and definitions:

Support functions. This section contains mathematical definitions of how transformation matrix objects are created from basic axis rotations and translations, inverted, composed with other transformation matrices, and converted to a vector representation using Euler angles for orientation.

Coordinates. The sizes of the vectors for robot, task, and uncertainty coordinates are defined.

Objects and rigid handles. Based on the coordinates defined above, in the next section the coordinate systems describing the robot, tool, and workpiece objects are defined. The object description currently includes the base frame of the object and a list of handles, which are frames defined relative to the base frame and used to connect the object to other components. For instance, the feature frames are defined as handles. The direct robot kinematics is included explicitly in the robot object, through the inclusion of the flange frame o_2 as a handle, the position of which is defined on symbolic form relative to the robot base.

Joints and kinematic loops. The objects are interconnected at the handles through generalized joints, which are just coordinate-dependent transformations describing the position and orientation of one handle with respect to a handle on a different object. Kinematic loops are also defined, which are just an ordered array of references to handles and joints, describing which transformations are part of the closed kinematic loops for each feature.

Output/measurement equation. The outputs, or equivalently the measurements, are defined as a function of the task and robot coordinates.

Automatic code generation. Code is generated for computing the all the parametrized rigid transformations defining the position of objects and handles. The position loop constraint functions l can be composed in a standardized fashion from these auto-generated functions, together with the previously defined kinematic loops. Likewise, C functions for the output and measurement equations are also generated automatically.

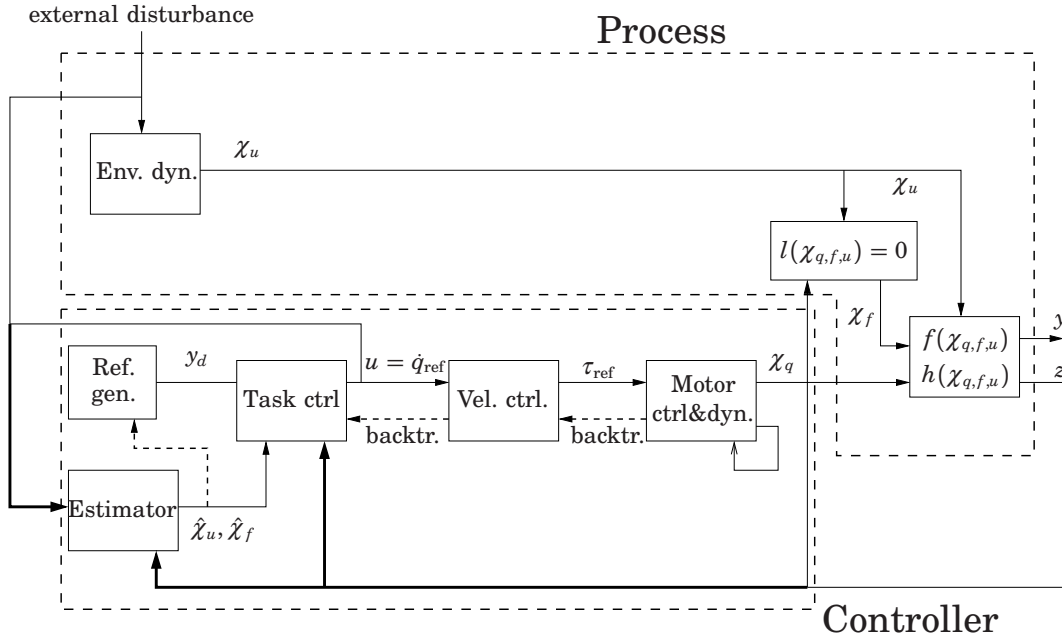


Figure 6 Suggested structure for a control system based on the constraint-based task specification.

The resulting files are the mex functions `lfunc_sfn.c` and `hfunc_sfn.c`, as well as a number of support functions used to setup initial values in the simulation.

The controller is implemented in the Matlab script `run_control.m`. The script runs a sampled representation of the controller, using the generated functions for generation of the Jacobians used in the controller. In addition, a number of other Matlab function are necessary. The function `compute_ref.m` is used to calculate the reference trajectories by simulating the manual teaching procedure given a specified end-effector position. Function `compute_feat_coord.m` is used in the simulation of the real process in order to compute the true values of the task coordinates χ_f , without performing numerical iterations as is done in the controller. `l_func.m` and `h_func.m` are Matlab implementations of the position loop constraint and measurement equation, which can be used for testing purposes. These functions in turn use a number of functions from the robotics toolbox for Matlab [2].

Control Structures

A suggested control structure can be seen in Fig. 6. In the general case, the controller implementation would contain the following blocks:

Reference generator. The generation of reference trajectories for the outputs should be produced in a separate block. This block needs access to the entire task description, as the trajectories must satisfy both dynamic and position loop constraints. A further complication is that the feasibility of the generated trajectories depend on the values of the uncertainty coordinates, which may be time-varying. In the example, this problem is solved by assuming that the trajectories have been obtained by teaching using *constant* uncertainty, so that trajectories are always feasible with respect to the position loop constraints. In the general case, it should be possible to update the trajectories online based on the estimated uncertainty, as indicated by the dashed line from the estimator output in Fig. 6.

Motor and velocity control. These blocks represent the low-level control at the joint level, typically run at a sampling rate of at least an order of magnitude faster than the outer task control. The structure of the outer blocks do not put any special requirements on the low-level control, and these blocks can therefore be designed independently from the task description, or even by re-using existing built-in servo controllers.

State estimator. The full state of the system consists of task and uncertainty coordinate vectors and any dynamic states, should be estimated. In most controllers, at least some of the states required by the controller can not be measured, and must be estimated. In the example, even though all outputs are measurable, the full state is required for computation of the local Jacobians of functions l and h . Also this block is dependent of the task specification, as care must be taken to generate estimates which are consistent with the position loop constraints. The original paper [3] suggests an estimator structure with separate update and correction steps as in a Kalman filter, where in both steps estimates are made consistent such that $l(\chi_q, \hat{\chi}_f, \hat{\chi}_u) = 0$. If χ_q is measurable, in practice the estimator only needs to estimate the uncertainty χ_u , as χ_f can be then be resolved from the position loop constraints. However, complete estimation of χ_u is often not feasible due to problems with observability, and it may be necessary to use reduced order estimators and nominal values for some states. Determining which states need to be estimated is a non-trivial problem, which most likely would require manual analysis in each given problem.

Task/position control. The structure of the controller itself is straightforward, given the task description, measurements, references, and state estimates. Jacobians of the functions l and h (or f) are generated numerically¹, and used to compute the effective gains (or linearization matrices) given the estimated coordinates. The controller generates a desired reference for the joint velocity control, which is sent to the inner loop. As the position control is just a (linearizing) proportional control, problems with integrator/state windup are avoided on this level.

3. Conclusions

The constraint-based techniques of [3] present a convenient and systematic methodology for modeling of complex sensor-based robot systems and tasks, such as in the setup introduced in this case study. The task specification is essentially based on position, which means that for instance force sensors are in general modeled by using spring-like relations to position coordinates. Extending previously designed system descriptions with extra sensors and feature frames is straightforward, and requires no modifications in the previous modeling. However, as stated also in [3], complete application of the methods down to the control level is quite involved in complex tasks. A further issue is that the task specification does not impose a clear and unique way to define relevant objects and feature frames, resulting in a possibility (or a requirement?) for user input during the task specification stage. User support through a software environment for task specification and optimization is proposed in [3], but the development of such an environment is the subject of further work. In such a software environment, partial task descriptions could be stored as templates in a library and assembled in a

¹Symbolic expressions for the Jacobians could in theory be computed and included in the code generation. However, in most problems this approach would be infeasible due to the size of the symbolic expressions involved, making numerical computation more efficient.

modular fashion, in order to speed up and simplify the user input. The result from the task modeling environment would be a file with the mathematical description of the specified task, stored in symbolic form in a portable format such as MathML.

Comments and Open Issues

Here some of the remaining open issues are summarized, and some of the choices made are discussed, together with alternative solutions.

Dynamic reconfigurability. In mobile robot systems, there must be a possibility for reconfiguration of the control as the robot encounters previously unknown areas of the environment. Other unmodeled events that need to be handled are sensor failures (e.g. occlusion or loss of tracking in a camera), or configuration changes at run-time. Similar situations occur also for many other types of robot systems, as different controllers with different parameters and object models may be needed, depending on the situations encountered during operation. Some of these events could be handled in a straightforward way by dynamically updating the values of the corresponding coordinate representations, or by introducing extra tunable parameters which could be set by the run-time control system. However, how to achieve a reconfigurable system in the general case is not clear, and depends strongly on the chosen control structure.

Control structure and implementation. The controller and estimation algorithm should be specified in a way that takes mathematical properties, implementation efficiency, proper modularization for flexible implementation, and real-time aspects into account. The implementation of the controller structure in Fig. 6 could be done using SDS blocks. The SDS (Sampled Dynamic System) blocks formalism presents an alternative approach. SDS blocks have properties permitting composition such that regulator objects or other sampled dynamic systems comprise entities with the same compositional properties as their aggregated parts. SDS blocks are mathematically causal, i.e. their outputs are well defined and depend on former and current inputs and internal states. The SDS blocks allow for backtracking of signals to the preceding blocks in the loop in a systematic way, which is necessary for instance to avoid windup in the outer control loops if saturations occur in the inner loops. On the mathematical level, the function of the blocks should preferably be specified in a non-causal, declarative way. Languages such as Modelica [1], could serve as an inspiration for this. However, there should also be possible to add optional causality specification, in order to support automatic code generation, and facilitate integration with for instance a causal computer-level representation using SDS blocks.

Control issues. Once the complete task description and specification is available, it is stated in [3] that the calculations for control and estimation can be easily automated. However, the paper suggests a control method for ideal velocity-controlled manipulators, based on controllers including linearization of the geometrical functions describing the task, and (Extended) Kalman filters for estimation of the potentially high-order task and uncertainty models. In the case study presented above, such an estimator would amount to simultaneous tool calibration, robot kinematics calibration, workpiece calibration, and camera hand-eye calibration, in addition to estimation of any dynamic states. In the presented example, such calibration would just not be possible to achieve in a robust and reliable way, even if the dynamic Kalman filters could be replaced by more suitable dedicated methods for static calibration.

Other options include the use of some type of reduced-order observers for estimation of only the required parameters. The properties of the presented controllers in the general case are not clear, as questions of observability, stability, and robustness are not addressed. In complex tasks (such as the type of tasks for which the method was developed), the proposed control design methodology would require extensive application-dependent tailoring on the control level, in order to be applicable to the given problem.

4. References

- [1] Modelica Association. Modelica. <http://www.modelica.org>.
- [2] P. Corke. Robotics Toolbox for Matlab. <http://www.petercorke.com/Robotics%20Toolbox.html>.
- [3] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbelien, K. Claes, and H. Bruyninckx. Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *International Journal of Robotics Research*, 26(5):433–455, 2007.
- [4] K. Nilsson, T. Olsson, and H. Bruyninckx. Basic robotics standards (BRoS)—motivations and examples. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, 2007. Submitted for review.