

Real Time Path Correction of Industrial Robots with Direct End-Effector Feedback from a Laser Tracker

Santiago Droll
New River Kinematics

ABSTRACT

In contemporary industries the demand for very accurate robots is continuously growing. Yet, robot vendors are limited in the achievable accuracy of their robots, as they have no means to provide a direct end-effector feedback. Therefore, most approaches aim to identify an accurate model of the robotic system, thus providing compensation factors to correct the deflections. Models, however, are unable to represent the real physical system in a sufficient manner for path correction. The non-linearities in robotic systems are difficult to model and the dynamics cannot be neglected. A better approach is, therefore, to use direct end-effector position and orientation feedback from an external sensor as, e.g. a Leica laser tracker. The measured data can directly be compared to the nominal data from the path interpolator. Hence, the data are independent of the kinematic robot model. The residual errors can be used to calculate correction values in Cartesian space, which are mapped to each individual robot joint, thus providing a fast path correction algorithm. The evaluation of this procedure has shown that it enables the robot to drive paths at a velocity of 100 mm/s with an RMS of only 0.11 mm. In this paper an algorithm, which is implementing this procedure is presented and evaluated on a iso path.

CITATION: Droll, S., "Real Time Path Correction of Industrial Robots with Direct End-Effector Feedback from a Laser Tracker," *SAE Int. J. Aerosp.* 7(2):2014, doi:10.4271/2014-01-2248.

INTRODUCTION

The demand of robot capabilities for accurate drilling, milling, welding or assembling on a large scale is continuously growing. Such tasks are highly dynamic and need a sophisticated control system in order to achieve the required accuracies and forces. Contemporary industrial robots are incapable of providing this accuracies because of imperfections in the encoder pinion gears, structural deformations and errors in the joint links. In order to achieve high path accuracies in position and orientation, a real-time control of the end-effector is needed. The therefore necessary controller can be implemented as a compensation algorithm, which is using external data feedback from a laser tracker.

Compensation Algorithm

The compensation algorithm is based on a linear controller that uses measurement data from an external sensor and nominal data from the robots path interpolator. For the data processing the sensor data needs to be in the same coordinate frame as the robot. The nominal path data are directly retrieved from the robots path interpolator and compared to the measured data from the sensor. The thereof calculated residuals are passed to a linear controller, which returns corrections in Cartesian coordinates. These correction values are subsequently mapped to the robots joint space by an inverse Jacobian algorithm. The Jacobian algorithm calculates an analytic

Jacobian matrix based on the robots current joint values and its DH-parameters. This matrix is then inverted and multiplied by the Cartesian corrections vector, hence, resulting in joint correction values. The correction values are summed up with the nominal values from the robots path interpolator and provided as input to the robots joint controllers (see [figure 1](#)).

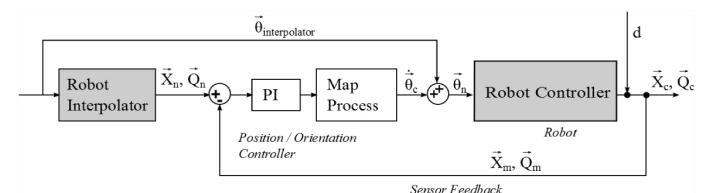


Figure 1. Data Flow in Controller

METHODS

The data from the external sensor are recorded as three spherical coordinates for the position and a quaternion for the orientation, all relative to the sensors base system. As the tracker and the robot have not natively the same base coordinate frame, a calibration is necessary to identify the transformation between the systems. The calibration can be done by measuring identical points in both systems and then correlating them with a best-fit alignment. However, it has to be considered that the robots measurements are less accurate

than the sensors, as they depend on the robots kinematic model. Therefore, it is desirable to use a sophisticated calibration method, which takes the robots kinematic model into account. Another method would be to identify the robots base frame directly via geometric relationships or the manufacturers' specifications.

For this paper an identification over identical points was chosen. Nineteen different end-effector measurements, distributed over the robots range of motion were recorded. Along with the end-effectors position and orientation the robots' joint angles were stored. Those measurements were then processed by an optimization algorithm which identifies optimized DH-parameters for the kinematic model and an optimized homogeneous transformation matrix for the robot-sensor transformation. A more precise description of the calibration process can be found in [ZR87].

The controlling algorithm itself can be split into three major parts. The data transformation and error calculation of the controller, the calculation of the Jacobian matrix and the joint space mapping.

Controller

To implement the controller algorithm, first, the data have to be transformed. They have to be presented in the same reference frame and notation. For the orientation of the data the quaternions notation was chosen because of its unique description in space and because of the simple conversion to the axis angle notation, which is important for the joint space mapping later on. It is important because the correction algorithm tries to turn all three axis simultaneously, whereas euler angles turn each axis consecutively. Therefore, euler angles would deliver different values, which would cause the algorithm to diverge.

First, the measured quaternion has to be transformed to a rotation matrix. This is necessary because quaternions are bound to their base system and cannot easily be transformed to another base system. Therefore, the measured quaternion is transformed to a rotation matrix, multiplied by the sensor-robot transform and then backwards transformed to quaternion notation. This way it is guaranteed that the nominal and the measured quaternion are represented in the same coordinate frame. The nominal robot data are transformed to quaternion notation as well (see [equation \(1\)](#)). Position data are already in the correct notion. Therefore the measured position data set has only to be multiplied by the robot-sensor transform. In [equation \(1\)](#) $\vec{X}$ is the position vector, $\vec{Q}$ is the quaternion and H_S^R is the sensor-robot transformation matrix.

$$\vec{X}_{meas} = H_S^R \cdot \vec{X}_{meas} \quad (1)$$

$$\vec{Q}_{meas} = f_{\rightarrow Q}(H_S^R \cdot f_{\rightarrow M}(\vec{Q}_{meas})) \quad (2)$$

$$\vec{Q}_{nom} = f_{\rightarrow Q}(\vec{\Phi}_{xyz_{nom}}) \quad (3)$$

If the robot-sensor transformation is known and all data are in the same notation, the residual errors can be calculated. The position residuals are calculated by subtracting nominal and measured data. The residual quaternion, is the quaternion that turns the measured quaternion into the nominal quaternion. This quaternion is calculated by multiplying the inverse of the nominal quaternion with the measured quaternion [Mac].

$$\epsilon_{\vec{X}} = \vec{X}_{nom} - \vec{X}_{meas} \quad (4)$$

$$\epsilon_{\vec{Q}} = \vec{Q}_{nom}^{-1} \cdot \vec{Q}_{meas} \quad (5)$$

In [equation 4](#) $\vec{X}$ is the position vector and $\vec{Q}$ the quaternion of the end-effector, ϵ is the vector containing the residual errors.

The residuals are then amplified by a linear PI controller which returns correction values in Cartesian space. The I-part integrates past values, thus reducing the steady state error to zero, whereas the P-part proportionally amplifies the current error. The behavior of the controller can be steered by the factors K_P , which is the proportional gain and K_I , which is the integral gain (see [equation \(6\)](#)).

$$\vec{Z} = K_P \cdot \vec{\epsilon} + K_I \cdot \int \vec{\epsilon} dt \quad (6)$$

In [\(6\)](#) $\vec{Z}$ is the vector containing the Cartesian corrections and $\vec{\epsilon}$ is the vector containing the residual errors from [equation \(4\)](#) (after the transformation to axis angles). However, these corrections cannot directly be applied to the robot. They represent the necessary changes in the end-effector position/orientation, but the robot is a kinematic chain of rotational joints, which cannot directly apply those values. Therefore the corrections have first to be mapped to joint space, before they are applicable to the robots' joint controllers. For this mapping an inverse Jacobian algorithm, which provides a local mapping is used.

Jacobian Algorithm for Joint Space Mapping

The kinematic configuration of the robot, which is a kinematic chain, introduces several limitations to the system. First of all, the joint space mapping has to be recalculated for every correction step, as the mapping is only approximated locally around the current position. Secondly, the kinematic configuration implies that a change in one joint will move all end-effector coordinates. Or in other words the system is tightly

coupled and highly non-linear. Therefore, it is questionable if a linear controller is the right choice for path correction, as it “sees” only one coordinate at a time and does not realize its influence on the other values. However, experiments have shown that the cross-coupling effects are small enough to be neglected, as long as the controller frequency is high enough and the path velocity does not exceed a certain level. The third challenge, the non-linear mapping can be solved by an efficient inverse Jacobian algorithm.

The Jacobian is the matrix that represents the effects of changes in joint space to changes in Cartesian space. Or in other words the partial derivative of the forward-kinematics, as can be seen [equation \(7\)](#). θ_1 to θ_6 are the joint variables of the robot, which are contained in $\vec{\theta}$.

$$J(\vec{\theta}, \dot{\vec{\theta}}) = \begin{bmatrix} \frac{\delta f_1(\vec{\theta})}{\delta \theta_1} & \frac{\delta f_2(\vec{\theta})}{\delta \theta_1} & \cdots & \frac{\delta f_n(\vec{\theta})}{\delta \theta_1} \\ \vdots & \ddots & & \vdots \\ \frac{\delta f_1(\vec{\theta})}{\delta \theta_m} & \frac{\delta f_2(\vec{\theta})}{\delta \theta_m} & \cdots & \frac{\delta f_n(\vec{\theta})}{\delta \theta_m} \end{bmatrix} \quad (7)$$

For a robot with six joints and six degrees of freedom, the Jacobian is a 6×6 matrix, where every row represents the influence on one Cartesian coordinate. As it is very cumbersome to calculate the derivatives it is mostly done numerically. However, as for real-time correction time is a limiting factor, an analytic approach to calculate the Jacobian was developed.

Jacobian Matrix

Using general mechanics, the velocity of a point turning around an axis can be described as the crossproduct of the points distance vector and the angular velocity plus the linear velocity [MBS] (see [equation \(8\)](#)). In [equation \(8\)](#) $\vec{v}$ is the velocity of the point, where $\vec{v}_{linear}$ is the linear velocity, $\vec{\omega}$ is the angular velocity and $\vec{r}$ is the distance vector to the point.

$$\vec{v} = \vec{v}_{linear} + \vec{\omega} \times \vec{r} \quad (8)$$

As industrial arm robots mostly have revolute joints, the linear velocity will be zero. The Jacobian can therefore be constituted as follows.

$$J(\vec{\theta}, \dot{\vec{\theta}}) = \begin{bmatrix} \vec{\omega} \times \vec{r} \\ \vec{\omega} \end{bmatrix} \quad (9)$$

$$J(\vec{\theta}, \dot{\vec{\theta}}) = \begin{bmatrix} J_v(\vec{\theta}) \\ J_\omega(\vec{\theta}) \end{bmatrix} \dot{\vec{\theta}} \quad (10)$$

According to theoretical mechanics, the angular velocity of the end-effector can be separated in single terms. Each term represents the influence of one joint. Summing up all terms leads to the end-effector spatial velocity. Applying the DH-convention, all joints must turn around their z-axis. Therefore, the angular velocity can be rewritten to the following form.

$$\vec{\omega} = \left[R_1^I(\vec{\theta})\vec{e}_z, R_2^I(\vec{\theta})\vec{e}_z, \dots, R_n^I(\vec{\theta})\vec{e}_z \right] \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_n \end{bmatrix} \quad (11)$$

In [equation \(11\)](#) R_1^I is the rotation matrix, which is transforming the i-the coordinate frame to the inertial system. $\vec{e}_z$ is the normal vector along the z-axis of the coordinate frame and $\dot{\theta}$ is the angular velocity of the corresponding robot joint. Comparing this notation to [equation \(9\)](#) shows that it is identical. It is the angular velocity Jacobian multiplied with the joint angular velocities.

$$J_\omega(\theta) = \left[R_1^I(\vec{\theta})\vec{e}_z, R_2^I(\vec{\theta})\vec{e}_z, \dots, R_n^I(\vec{\theta})\vec{e}_z \right] \quad (12)$$

Recalling [equation \(8\)](#) the spatial velocity can be written as cross-product of the angular velocities and the vector to the TCP. Using the definition of the angular velocity in [equation \(11\)](#), the linear velocity can be developed to the following form.

$$\vec{v} = \left[R_1^I(\vec{\theta}) (\vec{e}_z \times \vec{r}_1), \dots, R_n^I(\vec{\theta}) (\vec{e}_z \times \vec{r}_n) \right] \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \vdots \\ \dot{\theta}_n \end{bmatrix} \quad (13)$$

Comparing this notation to [equation \(9\)](#) shows, that it is identical. Therefore the linear velocity Jacobian can be written as follows.

$$J_v(\vec{\theta}) = \left[R_1^I(\vec{\theta}) (\vec{e}_z \times \vec{r}_1), \dots, R_n^I(\vec{\theta}) (\vec{e}_z \times \vec{r}_n) \right] \quad (14)$$

Thus, the Jacobian for the joint space mapping is identifiable by simple mechanics and geometrical relationships.

It can now be used to calculate a local mapping from Cartesian space to robot joint space.

Joint Space Mapping

As the Jacobian natively represents the influence of changes in joint space to changes in Cartesian space it has to be inverted for the use in the correction algorithm.

