Iterative Learning Control for Milling with Industrial Robots in Advanced Manufacturing

Thomas Daun

LUND UNIVERSITY

Department of Automatic Control

Msc Thesis
ISRN LUTFD2/TFRT--5942--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck
Lund 2014

Abstract

Iterative Learning Control for Milling with Industrial Robots in Advanced Manufacturing

The demand of today for advanced manufactured parts with high precision is due to the increasing complexity of technologies. The parts are typically made by CNC (Computer Numerical Control) machines, which are expensive and comparably big. By using industrial robots that are significantly cheaper, reduced costs can be achieved, which is particularly beneficial for small and medium enterprises. Robots are, however, less stiff and strong and are less accurate compared to the CNC machines.

In this thesis, the idea was that by designing a controller, this could be compensated for so that the robot could perform machining with high precision. The research made in this thesis was part of the EU co-funded research project COMET.

The robot task was to repeatedly mill parts with improved results. Iterative Learning Control (ILC) was therefore chosen as a suitable control strategy for this task. It uses the position error from previous iterations and adds it to the control signal to converge the output to successful results. Results showed that when using ILC for tracking paths, the position error could be reduced with approximately 11-20% in x, y, and z directions after one iteration.

Acknowledgment

First of all I would like to thank my supervisors Dr Anders Robertsson and Björn Olofsson for giving me support and feedback during this master thesis. I would also like to thank Dr Klas Nilsson for arranging, organizing and planning the part of the research that was done in Sheffield, UK. Thanks to Olof Sörnmo for helping me with the Simulink models and ExtCtrl. Dr Rolf Johansson, thank you for the feedback on the manuscript for this thesis.

Secondly, I would like express my gratitude to everyone who has helped me during my stay in Sheffield, UK. Thanks to Oliver Bailey and Robert Leach at AML Ltd, Sheffield for all your support and long days spent in the robot cell. I owe Roland Krain at TEKS Ltd great thanks for all the help from the very first day in Sheffield.

This work was supported by the ELLIIT Excellence Center at Lund University and the European Union's seventh framework program (FP7/2007-2013) under grant agreements COMET (Ref. #258769) and SMErobotics (Ref. #287787).

Contents

1. Introduction **7**
- 1.1 Background . . . 7
- 1.2 Problem Definition . . . 8
- 1.3 Experimental Setup . . . 9

2. Theory **15**
- 2.1 Theory of Kinematics . . . 15
- 2.2 Theory of ILC . . . 18
- 2.3 The ILC Problem Formulation . . . 18

3. Kinematic Model **21**
- 3.1 Verification of Kinematic Model . . . 23

4. Experimental Preliminaries **25**
- 4.1 Loss of Data During Logging . . . 26
- 4.2 Trajectory vs. Path Tracking . . . 27

5. Method **56**
- 5.1 ILC Script . . . 56
- 5.2 Creating ILC Path . . . 62
- 5.3 ILC Milling . . . 68

6. Results **76**
- 6.1 Only First Iteration ILC . . . 76
- 6.2 Free Space Results . . . 77
- 6.3 Milling Results . . . 79
- 6.4 Results from SIR . . . 91

7. Discussion **93**
- 7.1 Results in Relation to Problem Definition . . . 93
- 7.2 Possible Improvements and Error Analysis . . . 93

8. Conclusion and Future Work **96**
- 8.1 Conclusion . . . 96
- 8.2 Future Work . . . 97

A. Simulink Models **98**
A.1 Kinematic Model 98
A.2 Trajectory Compensated Model 98
B. RAPID Programs and Matlab Scripts **101**
B.1 Original RapidBox Program 101
B.2 ILC Script 103
Bibliography **121**

1

Introduction

Technologies of today require high precision manufactured parts to increase efficiency and performance in industries such as aerospace. These advanced manufactured parts are typically produced by milling with CNC (Computer Numerical Control) machines. They produce complex parts that require high tolerances with significantly less time and effort than manual milling. CNC machines, however, are expensive and comparably big, taking a lot of floor space. They also have limitations regarding the size of the part that can be machined.

By using industrial robots that are significantly cheaper, costs can be reduced, which is particularly beneficial for small and medium-sized enterprises (SMEs). However, robots are not as stiff or accurate as CNC machines. The idea for this thesis was that by using feedback control, this could be compensated so that the robot could perform machining with high precision.

1.1 Background

Milling with industrial robots in advanced manufacturing is a cost effective solution due to the fact that robots are significantly cheaper than CNC machines [Berselli et al., 2013a] and can use a smaller operating space. If mounted on rails, robots can also be used to machine larger parts. A robot also offers a more flexible solution since the same robot can be used for various tasks, both handling and machining, such as assembling of components, spray painting, and milling. Another benefit of using robots is that SMEs are more likely to afford a robot than a CNC machine and can therefore start producing parts after having invested in a robot. When the company grows it can gradually install more robot cells in order to increase production.

This master thesis was part of an EU co-funded research project called COMET, which started in 2010 and was a collaboration between the Robotics Lab at Lund University and other academic and industrial partners. The aim of the COMET research was to develop plug and produce components and methods for adaptive control of industrial robots enabling cost effective, high precision manufacturing in factories of the future [COMET, 2013].

Research shows that when milling surfaces with industrial robots, the accuracy can be improved by using control systems. With a joint stiffness model based feed forward approach, an industrial robot could be made more accurate when milling a surface in aluminum [Wang et al., 2009]. The challenge for this control solution was to simplify the stiffness identification process when creating the joint stiffness model. As a part of the COMET project, a macro/micro manipulator configuration was tested for milling surfaces in aluminum [Olofsson, 2013]. The milling experiments were performed on three different surfaces along a straight path, which individually tested and evaluated the results of face milling along the x, y, and z direction. Experimental results showed that the proposed actuator configuration, combined with the control architecture could be used for increasing the accuracy when milling faces. These results show the potential of using robots for milling in advanced manufacturing.

A robot cell was set up at AML Ltd in Sheffield, UK, one of the COMET partners and was used for the experiments and evaluation described in this thesis. Experiments based on the methods described in this thesis were also done on a similar robot cell by SIR in Modena, Italy which was one of the other COMET partners.

1.2 Problem Definition

Milling with industrial robots in advanced manufacturing is beneficial for industries and SMEs in particular. A cost effective solution, for high precision manufacturing, is successful if it can produce parts within acceptable time and tolerances. The system must also be time and cost effective regarding implementation and adaptation if it is to be industrialized and commercialized.

In an early stage, the COMET project only focused on evaluating if industrial robots could be made accurate enough when milling by using a control strategy that compensate dynamic position deviations. The aim of the research was, however, to have a control strategy that could simultaneously perform 3D-milling within acceptable tolerances. An important milestone for the project to move on to stages that focused on industrializing the solution. The time aspect and cost regarding production and easy implementation, as well as adaptation was therefore not considered in this part of the project.

This master thesis evaluates the possibility to reach acceptable tolerances by designing and implementing a suitable control strategy that improves the robot's performance when simultaneously milling in 3D. Compared to CNC machines, robots are neither as strong nor stiff and have less absolute accuracy. These deficiencies need to be considered and compensated when milling with robots. In this project it was decided that an acceptable tolerance was a maximum position error of ± 0.1 mm for milling [Berselli et al., 2013a]. It requires high-accuracy sensors integrated in a control system that can compensate dynamic position deviations.

The Iterative Learning Control (ILC) algorithm was chosen as a suitable control strategy since the robot should repeat the same task with increased accuracy of the results. In previous research, ILC has been proven to considerably improve the tracking accuracy of a laser welding head, mounted to an industrial robot [Hakvoort et al., 2007]. Conceptually, ILC uses the position error from previous iterations and adds it to the control signal. It is an off-line control method based on the assumption that the position errors that occur when milling were repetitive. The theory behind ILC is further described in Section 2.2.

To summarize, the problem definition for this master thesis can be formulated by the following requirements. These requirements were evaluated in this master thesis and they were important for the success of the COMET project at this stage.

- Perform high precision manufacturing by using industrial robots.
- Implement off-line control system with integrated, high accuracy, sensors.
- Design and implement an ILC control strategy that simultaneously improves the robot's accuracy successively in 3D when milling.
- Implement algorithms for milling with ILC in 3D.
- Milling result within specified tolerance of a maximum position error of ± 0.1 mm.

1.3 Experimental Setup

This section gives an explanation of methods used for this research and the setup of the robot cell as well as software, protocols, and implementation used for the experiments, evaluation, and control method.

Robot Cell

The robot cell at AML was set up of an ABB IRB 6640 robot with a mounted work object, a Nikon Metrology K600 tracking system and a fixed mounted spindle. An overview of the robot cell is seen in Figure 1.1.

ABB IRB 6640 Robot and IRC5 Controller The robot used at AML was an ABB IRB 6640-205/2.75. This robot model is one of ABB's stronger and rigid robot models which uses the second generation motion control with increased path performance and has an average accuracy of ± 0.5 mm [ABB Robotics, 2010]. The robot model has a weight of 1320 kg and is suitable for heavy material handling up to 205 kg. It has the working range seen in Figure 1.2 [ABB Robotics, 2007].

An ABB IRC5 control cabinet was used to control the robot. This robot controller is ABB's fifth generation of robotic control cabinets and consists of a main

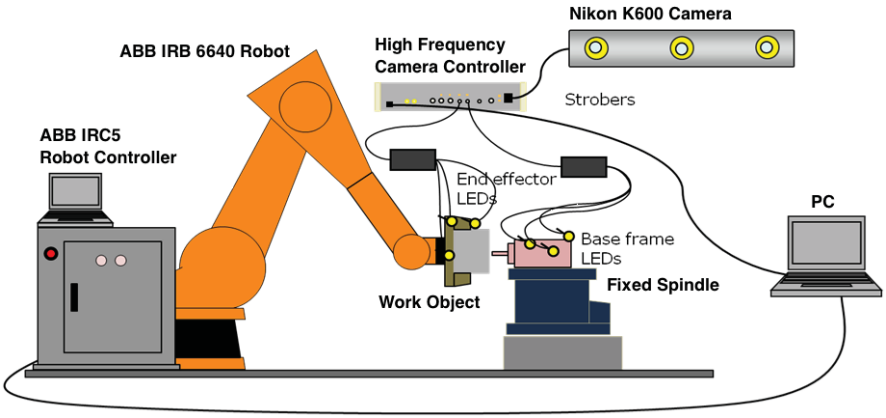


Figure 1.1 Overview of the robot cell at AML [Berselli et al., 2013a].

computer and an axis computer. The main computer includes path planning and trajectory generation from RAPID programs and manual jogging. In the axis computer the power feed to the robot motors are regulated. For more information the reader is advised to see [ABB Robotics, 2004] or [ABB Robotics, 2011].

Nikon K600 Tracking System and High-Frequency Controller The Nikon K-series tracking system measure positions of infrared LEDs and through triangulation calculate the 3D position of each LED [Nikon Metrology, 2013]. In this case the LEDs were mounted to the spindle and the robot's wrist. Measured positions were used for motion analysis and 3D inspection. An image illustrating a tracking system measurement of an LED is shown in Figure 1.3.

The K600 tracking system is used for high-frequency measurements. It uses a high-frequency (HF) controller with a sample rate of 250 Hz to acquire data. The Nikon software used was KLink with dMM (dynamic Motion Measurement), a digital metrology system that accurately tracks point coordinates and orientations of the LEDs. This software is used for motion and deformation measurement evaluation [Nikon Metrology, 2010].

Model and Control System

The model used for implementing the control strategy was designed to handle the signals sent to and from the robot controller, as well as the signals received from the tracking system. By processing signals into position coordinates with forward and inverse kinematic blocks, which is further described in Section 2.1, the signals could be interpreted and analyzed as positions in cartesian coordinates. These signals could be implemented in a controller designed within the model to control the robot. Following is a short explanation of the control system and software used for control implementation and code generation. For further details and information

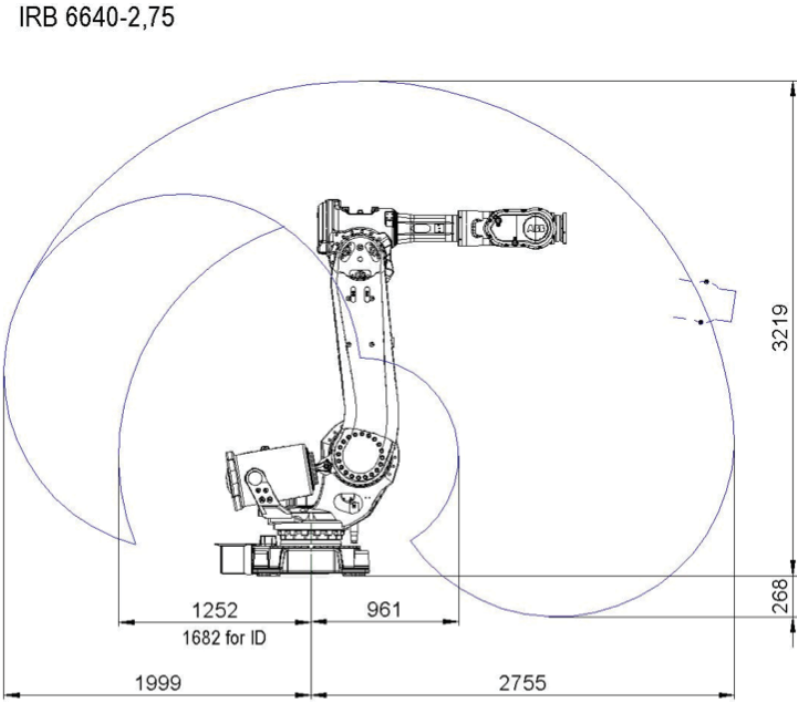


Figure 1.2 Working range in mm of the ABB IRB 6640-2.75 [ABB Robotics, 2007].

the reader is advised to read [Blomdell et al., 2010], [Dressler, 2012] and [Dressler, 2009].

Robot Control System The robot control system that was used in this thesis was extended to connect with computers for low level sensors. Figure 1.4 gives an overview of the extended control system. An extra module was connected to the ABB IRC5 controller and could be used to either read signals sent between the main and axis computer (submit) with a sample period of 4 ms or take control over the robot (obtain). This module controller runs on a Xenomai Linux PC and was implemented in the software Matlab with Simulink, which was then translated into C-code using Simulink Coder [Dressler, 2012].

Matlab, Simulink and Real Time Workshop The software used to implement the controller was Matlab with Simulink version 2009b. Commonly used blocks, S-functions and embedded Matlab code blocks are supported by the Matlab Real Time Workshop (RTW). They were used to implement complex relations such as forward and inverse kinematics. All signals could be set up in the model to be logged as well as the parameters that should be tuned in a GUI (Graphical User Interface). With

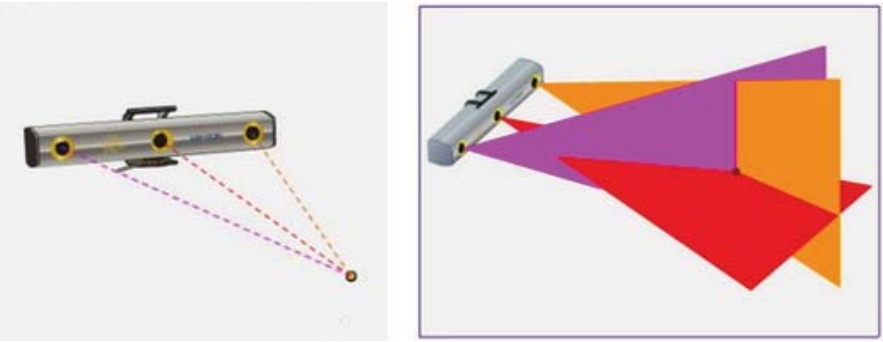


Figure 1.3 Measurement with Nikon K600 tracking system [Nikon Metrology, 2013].

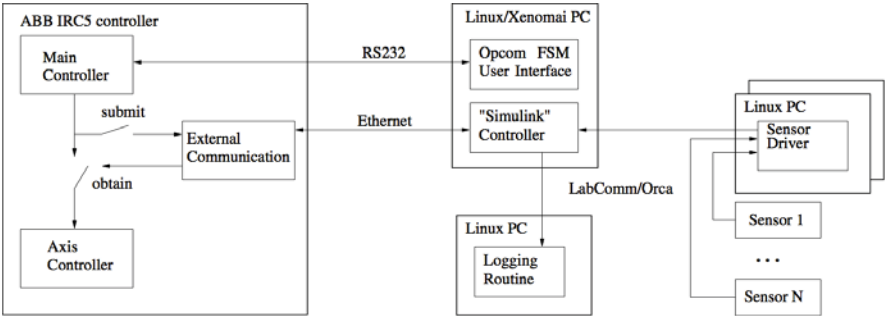


Figure 1.4 Control structure of the extended ABB IRC5 control system [Dressler, 2012].

Matlab RTW the model and controller could be translated into C-code, which could be compiled and loaded for control [Blomdell et al., 2010].

ExtCtrl Library To transform coordinates between different coordinate systems, the Simulink blocks available in the ExtCtrl library were used. There are different blocks for the IRB robot in this library and they were used for joint angles expressed on arm or motor side. The translation from motor to arm angles was done with the *motor2arm* block and the *arm2motor* block translated the arm angles into motor angles. All matrices are given as arrays where the elements are stored row after row [Dressler, 2009]. In the ExtCtrl library there are also Quaternion, T44 conversion, forward and inverse kinematics blocks available, further explained in Section 2.1. An overview of available blocks is given in Figure 1.5.

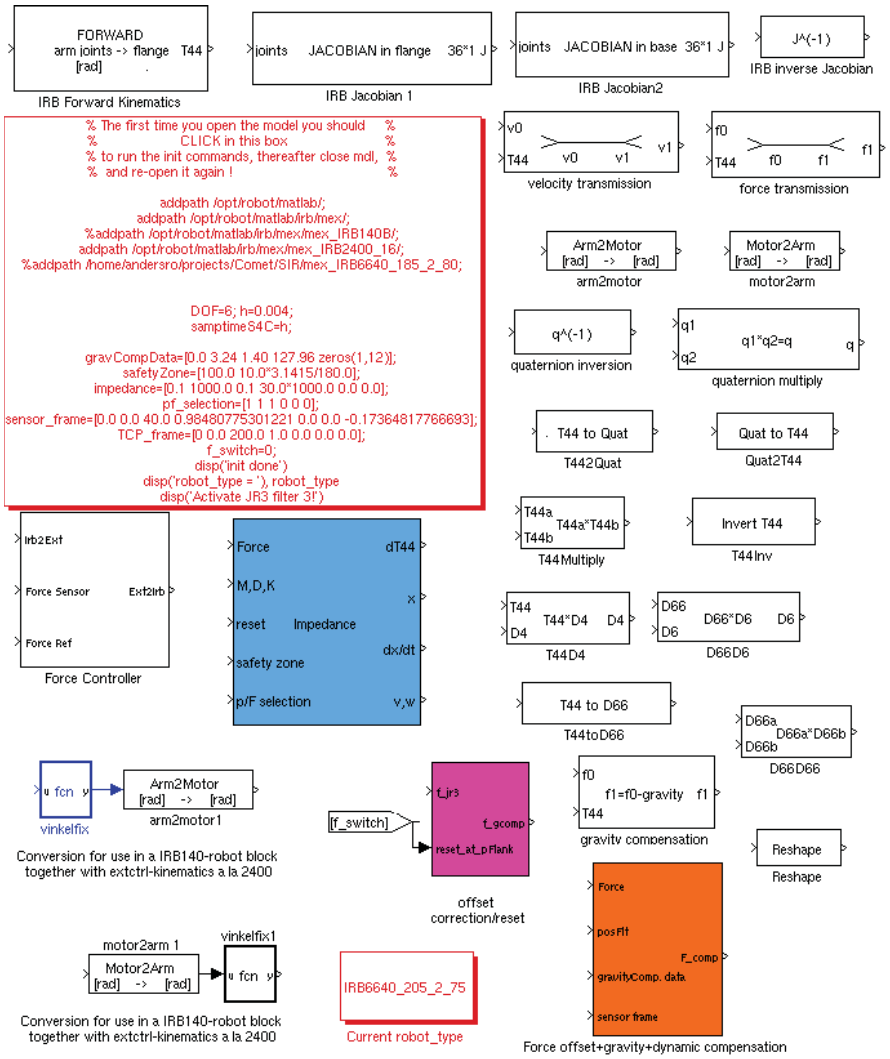


Figure 1.5 Overview of the Simulink blocks available in the ExtCtrl library.

Network and Data Communication

This is a short explanation of the protocols and programs used for network and data source communication to sample data streams and interpret them. The readers is advised to read [Blomdell et al., 2010] for further details and information.

LabComm The binary protocol LabComm was used for the communication between software and sensor data with different data sources and samples. It synchro-

nizes data sources and samples and verifies that all samples have same identical layout as the sample data. Various samples such as primitive types, fixed, variable size and multidimensional arrays as well as arbitrary nested structures can be sent with LabComm. A LabComm data stream can be stored to a log file which can be interpreted with a separate program.

ORCA and Opcom ORCA is a protocol built on top of LabComm for two-way communication, which divides samples into inputs, outputs, parameters, and log signals. Opcom is a GUI for ORCA, used to load models built in Simulink. The parameters that were set up in the model could be accessed and tuned in Opcom.

rtw2orca Code generation for the robot controllers was done with Matlab/Simulink RTW. To integrate with software that might be required for executing the controller, the tool rtw2orca was used. It uses a template file together with LabComm files to generate a main program that handles data between protocols and code generated by RTW.

Kst For viewing and plotting the data, Kst was used. It is a tool for viewing and plotting real time large data set with built-in data analysis functionality [KDE, 2013].

Research Approach

To systematically perform experiments and evaluate the system and the performance of the robot, the research was divided into steps:

- Build a model in Matlab, Simulink with ExtCtrl library to log the ABB robot and Nikon tracking system signals.
- Verify the kinematics setup in the model by modifying signals into Cartesian coordinates using forward and inverse kinematic blocks.
- Modify reference signals sent from the robot and tracking system into the same coordinate system to evaluate the performance of path tracking.
- Analyze the error between desired robot position and the measured data from the tracking system to determine performance behavior caused by disturbances, such as vibration.
- Design an Iterative Learning Control (ILC) algorithm to reduce errors.

2

Theory

This chapter describes the theory that this thesis was based on. It describes the theory behind robot kinematics and the Iterative Learning Control (ILC) strategy.

2.1 Theory of Kinematics

In order to control a robot and evaluate its performance, it is important that the data received for interpretation is correct. To achieve correct data it requires an accurate kinematic model where the dimensions of the robot are specified so that its movements can be transformed into coordinates in a reference frame. With a correct representation of the coordinates the data can be analyzed and used as inputs for a controller.

This section describes the theory behind the transformations of coordinates and orientations in space used in the Simulink model. The following theory and equations of kinematics are found in [Spong and Vidyasagar, 1989]. A robot manipulator consists of a kinematic chain of links connected with joints. To control a robot the operator needs to know the position of the end-effector or tool in a certain coordinate system. Typically, a robot can give information about the angles of its joints on the motor side, but not of the end-effector. The purpose of kinematics is to determine the tool position and orientation by knowing the angles of the joints. With information about the angles the position and orientation of the tool can be calculated, which is called the forward kinematics problem. The inverse kinematics problem is the inverse calculation where the angles of the joints are calculated from a desired position and rotation of the tool in a cartesian coordinate system.

Homogeneous Transformations

The establishment of various coordinate systems that represent positions and rotations of rigid objects is important in robot kinematics. With these coordinate systems, translations among them are used to calculate their positions relative to the reference frame. To have a general reference it is customary to establish a fixed coordinate system called the *world* or *base* frame. Translation and rotation matrices

are used to describe these relations. The homogeneous transformation matrix H, given as

$$H = \begin{bmatrix} R & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.1}$$

is a transformation matrix that combines the two operations into one, where R is the rotation matrix, $\mathbf{d}$ the translation distance and $\mathbf{0}$ denotes $(0,0,0)$. The transformation matrix T, given as

$$T = \begin{bmatrix} n_x & s_x & a_x & d_x \\ n_y & s_y & a_y & d_y \\ n_z & s_z & a_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{s} & \mathbf{a} & \mathbf{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

is considered the most general one, where $\mathbf{n}$, $\mathbf{s}$ and $\mathbf{a}$ are vectors representing directions of axes in relation to the coordinate system and $\mathbf{d}$ represents the vector between the origins of the coordinate systems.

Forward Kinematics

With given joint angles, the position and rotation of the tool is calculated relative to the reference coordinate system. Suppose a robot has $n+1$ links, numbered from 0 at the base of the robot to n and joints numbered from 1 to n [Spong and Vidyasagar, 1989]. Then the i-th joint, denoted as q_i is the point in space where links $i-1$ and i are connected. To each link a coordinate frame is rigidly attached with a frame at the base referred to as 0. The frames are chosen as 1 to n so that frame i is rigidly attached to link i. The idea of attaching frames rigidly to links is illustrated on an elbow manipulator in Figure 2.1.

Suppose now the matrix A_i is the homogeneous matrix that transforms coordinates of a point from frame i to frame $i-1$. The matrix is not constant but varies according to the joint variable q_i, which gives

$$A_i = A_i(q_i). \tag{2.3}$$

The transformation matrix that transforms coordinates of a point from frame j to frame i is then given by the homogeneous matrix T_i^j. It is given by

$$T_i^j = A_{i+1}A_{i+2}\ldots A_{j-1}A_j, \quad \text{if } i < j \tag{2.4}$$

$$T_i^j = \begin{cases} \mathrm{I} & \text{if } i = j \\ (T_i^j)^{-1} & \text{if } j > i \end{cases}. \tag{2.5}$$

According to the manner that the frames have been rigidly attached to the corresponding links, any point on the end-effector expressed in frame n is constant independent of the robot configuration. The homogeneous matrix (2.5) is given by

$$H = \begin{bmatrix} R_0^n & \mathbf{d}_0^n \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.6}$$

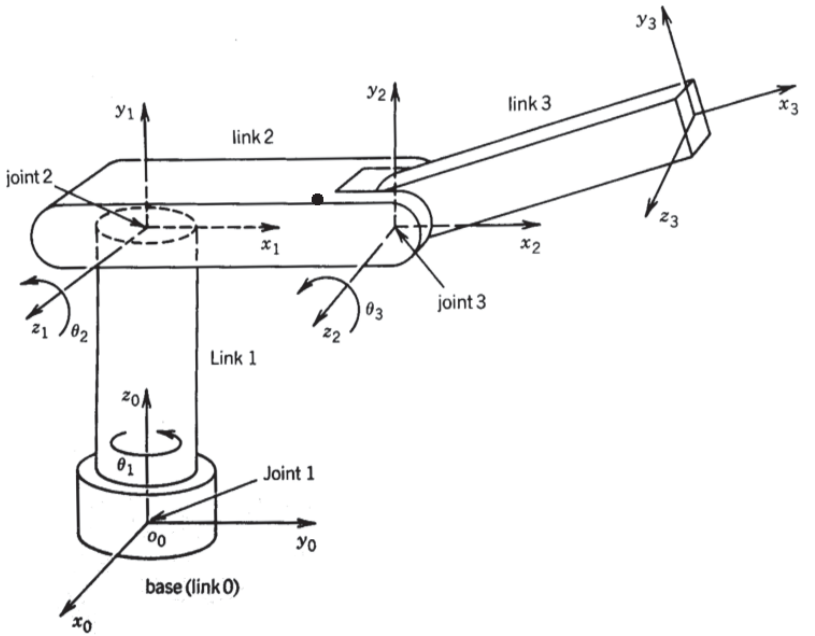


Figure 2.1 Coordinate frames attached to links on an elbow manipulator [Spong and Vidyasagar, 1989].

with a position and rotation of the end-effector with respect to the *base* frame, represented by the vector $\mathbf{d}_0^n$ and the rotation R_0^n. Derivation of position and rotation of the end-effector in the base frame is then done by

$$H = T_0^n = A_1(q_1)\dots A_n(q_n). \tag{2.7}$$

The form of each homogeneous transformation is given by (2.7) and

$$A_i = \begin{bmatrix} R_{i-1}^i & \mathbf{d}_{i-1}^i \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.8}$$

Finally this gives

$$T_i^j = A_{i+1}\dots A_j = \begin{bmatrix} R_i^j & \mathbf{d}_i^j \\ \mathbf{0} & 1 \end{bmatrix}. \tag{2.9}$$

Inverse Kinematics

If a robot should be controlled to move into a certain tool position, the joint angles need to be calculated. It gives the problem of finding joint angles for a desired end-effector position and orientation, which in general is more difficult than solving the forward kinematics problem because of the fact there are multiple solutions.

A certain end-effector position and orientation can often be reached with different joint angles.

With a given homogeneous matrix H, described in (2.1), the inverse kinematics problem is stated to find one, or all solutions to

$$T_0^n(q_1,\ldots,q_n) = H, \tag{2.10}$$

followed by

$$T_0^n(q_1,\ldots,q_n) = A_1 \ldots A_n. \tag{2.11}$$

From (2.10) follows 12 nonlinear equations with n unknown variables $q_1,\ldots,q_n$.

2.2 Theory of ILC

Iterative Learning Control (ILC) is a strategy to iteratively compensate repetitive errors. The concept of ILC is known to have been introduced by Uchiyama in 1978 [Uchiyama, 1978]. However, ILC is not considered to have ben an active research area until 1984, when an article by Arimoto, Kawamura, and Miyazaki [Arimoto et al., 1984] was published.

When machines are repeatedly doing the same tasks the idea is to use the knowledge from previous iterations of the task to reduce the errors in the following iterations [Norrlöf, 2000]. Experimental results from recent research show that a model-based ILC procedure considerably reduced the tracking error of a laser welding head, mounted to a six-axes industrial robot, because of the repeatability of tracking errors [Hakvoort et al., 2007]. In this project the machining tasks performed by the robot were repeated to produce parts with improved accuracy. The errors that occurred from using the robot were considered to be repetitive and hence the ILC strategy was tested. If the errors were repetitive, ILC should significantly reduce them to only leave errors caused by stochastic, non-predictive disturbances in the system.

2.3 The ILC Problem Formulation

The theory and equations described in this section are based on [Norrlöf, 2000], where the reader can find more information about ILC. The idea of ILC is to find a control signal that follows the desired reference as well as possible through an iterative procedure. Through iterations the control signal is updated to find the input that converges the output error into successful results.

The ILC problem formulation is to find the input to the system such that the output follows the desired trajectory as accurately as possible by using an iterative procedure, given a reference trajectory and a system [Norrlöf, 2000]. With an available model of the system, the optimal solution is to invert the model and add it to

the output of the inverted model and use it as an input, which will produce the desired output. Suppose a system should follow a trajectory with high accuracy from a given reference signal $r(t)$ over an finite time interval $[0, t_f]$. Figure 2.2 illustrates a system where $r(t)$ is considered to be the desired position of a joint on the robot arm and G_C a discrete time SISO model of the closed loop that consists of the robots joint and controller. The start position of the joint and the speeds of its movements are assumed to be the same every time the procedure is repeated.

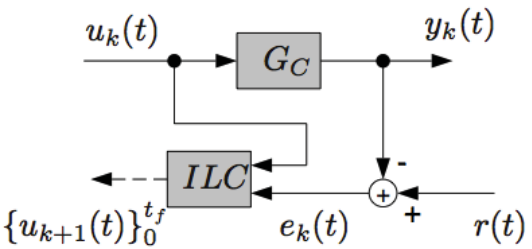


Figure 2.2 ILC controlled system [Norrlöf, 2000].

Initially the system is controlled with the signal $u_0(t)$, which gives the output result given by

$$y_0(t) = G_C(q)u_0(t), \tag{2.12}$$

where q is the time shift operator and the index 0 indicates how many times the iterative procedure has been repeated and is called the *iteration index*. The first time the procedure is performed the *iteration index* is 0. The difference between the desired and actual output, referred to as the tracking error, is then given by

$$e_0(t) = r(t) - y_0(t). \tag{2.13}$$

Assume that the initial condition is reset for every new iteration and that G_C is time invariant. The idea behind ILC is then introduced, where the tracking error will be the same for every iteration as long as the input signal is the same. With ILC algorithms an iterative search method to find the optimal input solution, given as $u(t) = G_C^{-1}(q)r(t)$ is introduced. This input solution gives the most common ILC updating equation given by

$$u_1(t) = u_0(t) + L(q)e_0(t), \tag{2.14}$$

where $L(q)$ is a linear discrete time filter that reduces high-frequency noise from the error signal before adding it to the input. Results from (2.12), (2.13) and (2.14)

gives the generalized ILC equations [Norrlöf, 2000] in

$$e_k(t) = r(t) - y_k(t) \tag{2.15a}$$
$$y_k(t) = G_C(q)u_k(t) \tag{2.15b}$$
$$u_{k+1}(t) = u_k(t) + L(q)e_k(t). \tag{2.15c}$$

These ILC algorithms are illustrated in Figure 2.2 as a block with $u_k(t)$ and $e_k(t)$ as input and the output sequence as $\{u_{k+1}(t)\}_0^{t_f}$. Following the algorithms in (2.15) is the transformation of the error between iterations given by

$$e_{k+1}(t) = (1 - G_C(q)L(q))e_k(t), \tag{2.16}$$

where the sufficient condition for the error to decrease is given by

$$|1 - G_C(e)L(e)| < 1. \tag{2.17}$$

The ILC algorithm given in (2.15) was implemented in the Matlab script described in Section 5.2 to generate ILC compensated tool paths.

3

Kinematic Model

A kinematic mode was created in order to control the robot. To evaluate a robot's performance it is important that the data received for interpretation are correct, which requires an accurate kinematic model. The kinematic model needs to be verified and the coordinate systems defined.

All the dimensions of the ABB IRB 6640-205/2.75 robot were implemented in a file that could be built into the Simulink model. The kinematic model was then verified by jogging the robot in Cartesian coordinates. If a correct model was built the Cartesian movements would only appear as movements in x, y, and z, respectively, when being plotted. Since this experiment only evaluated if the signals defined in the model were registered correctly in Cartesian coordinates, the choice of reference frame was not important. For this experiment the *flange* frame was used.

Coordinate Systems

To avoid unpredictable behavior of the robot it is important to keep in mind what coordinate system that is used. There are four common coordinate frames used to specify coordinates in relation to reference frames [Dressler, 2009]. The *base* frame is attached to the base of the robot and is the only constant one. On the flange of the robots end-effector is the *flange* frame. To define the tool center point (TCP) the *TCP* frame is used. For references in relation to the sensor, the *sensor* frame is used. The robots *base* and *flange* frames are shown in Figure 3.1.

T44 Conversion Block

To translate coordinates between reference frames the two blocks with *T44* conversions are available in the ExtCtrl library, inversion and multiply. If an input to a *T44* block fulfills

$$P_{base} = T44 \cdot P_{flange}, \tag{3.1}$$

where P_{flange} are the coordinates of point P given in the *flange* frame, the block output are the coordinates of P translated into the *base* frame [Dressler, 2009]. The

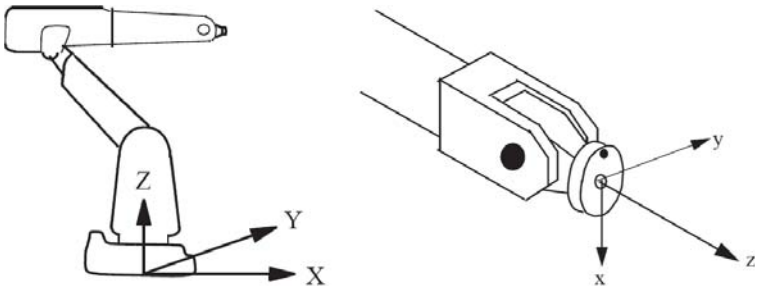


Figure 3.1 The robots *base* frame (left) and *flange* frame (right) [Dressler, 2009].

T44 inversion in

$$P_{flange} = T44^{-1} \cdot P_{base} \tag{3.2}$$

then translates coordinates from the *base* to *flange* frame. For the *T44* multiply the output of the block is the upper input multiplied with the lower one. If the upper input T_1 fulfills

$$P_{base} = T_1 \cdot P_{flange} \tag{3.3}$$

and the lower input T_2 fulfills

$$P_{flange} = T_2 \cdot P_{TCP}, \tag{3.4}$$

then the output $T_3 = T_1 \cdot T_2$ will fulfill

$$P_{base} = T_1 \cdot P_{flange} = T_1 \cdot T_2 \cdot P_{TCP} = T_3 \cdot P_{TCP}. \tag{3.5}$$

Quaternion Block

The ExtCtrl block such as *T44 to Quat* was used to translate the *T44* matrix into a representation with the translation vector and quaternion. This representation was given as (T, q_T), which has 7 elements with the first three representing the translation vector $T = (T_x, T_y, T_z)^{\mathrm{T}}$ and the remaining 4 quaternion elements represent the rotations q_T [Dressler, 2009] and [Dressler, 2012].

Simulink Model

The signal transformation of the robot joint positions into Cartesian coordinates is shown in Figure 3.2. First the signal was translated from motor angles into arm angles with the *motor2arm* block. The forward kinematics block *FORWARD* translated the joint angles into flange coordinates. In the last transformation block *T44 to Quat* the coordinates were translated into a (T, R_T) representation. A data stream *xyzTestAml* containing the x, y, and z values was plotted. An overview of the whole Simulink model used for kinematic verification built with the dimensions of the ABB IRB 6640-205/2.75 robot is shown in Appendix A.1.

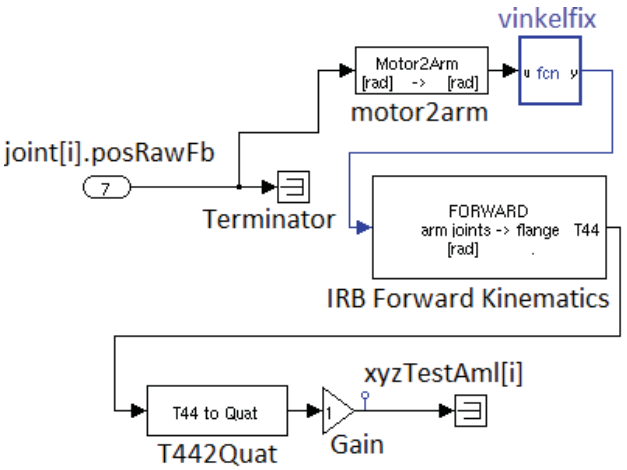


Figure 3.2 Signal transformation from motor angles to Cartesian coordinates.

3.1 Verification of Kinematic Model

By plotting the Cartesian positions over time as samples (*Point Index*) at 250 Hz in Kst, the kinematics could be verified. In Figure 3.3 the start position in x, y, and z at *Point Index = 0* was compared to the start positions shown on the ABB teach pendant in Figure 3.4. The robot was then manually moved in Cartesian directions starting with x, then y, and finally z. These movements only appeared in the corresponding diagram and no movements were registered in the other directions.

The results show that the kinematics built into the model were correct since the values of all start positions were the same and because the Cartesian jogging moves only appeared in the corresponding plot.

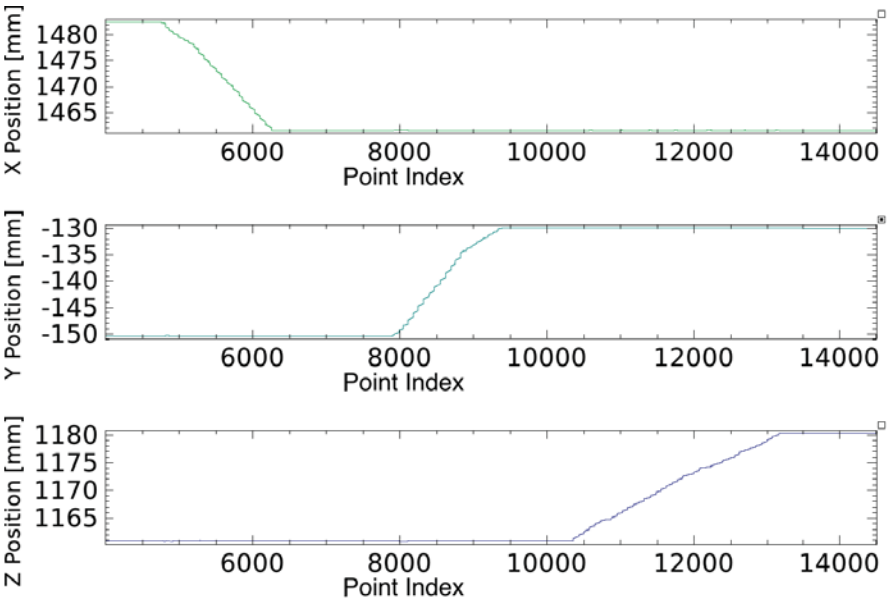


Figure 3.3 Robot positions in x, y and z plotted relative to samples (*Point Index*) [250 Hz].

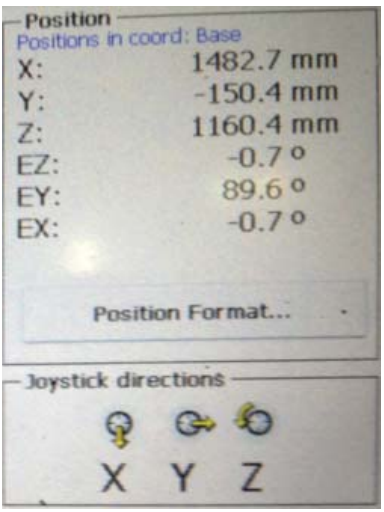


Figure 3.4 Start positions in Cartesian coordinates shown on ABB teach pendant.

4

Experimental Preliminaries

With a correct kinematic model, the robot's performance when tracking a reference path could be logged. A tool path created in PowerMill, which is a CAM (Computer Aided Manufacturing) software that can translate tool paths into RAPID code, was used as reference. The tool path was created to mill a pocket with a certain size and depth out of an aluminum block, referred to as the *Pocket* tool path. Figure 4.1 shows pockets that were milled by the robot using the PowerMill tool path. The reference sent to the robot and the positions measured by the tracking system were compared and analyzed to determine the robot performance.

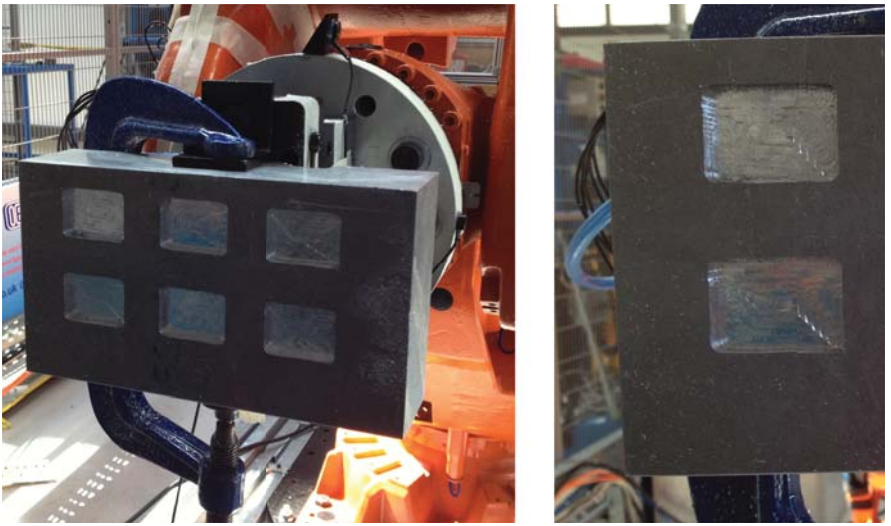


Figure 4.1 Pockets milled by robot using a PowerMill generated tool path.

4.1 Loss of Data During Logging

To analyze the performance of the robot, the logged x and z positions were plotted versus the y positions. The reference sent to the robot and the measured positions from the tracking system was then plotted similar to the *Pocket* tool path. When comparing the diagrams to the original reference it seemed like the robot was cutting corners while it was moving along the trajectory, which is shown in Figures 4.2 and 4.3.

To determine if the cutting corner behavior was caused by the robot controllers interpretation of the RAPID code generated by the PowerMill software, experiments with a simple RAPID code tool path were done. This tool path was created to generate a simple box, referred to as the *RapidBox* where the robot would move in a rectangle and then step down. This rectangle movement was repeated three times. The tool path was created in RAPID code by programming each corner of the box as points that the robot should move to and then move perpendicularly to the rectangle (step down). By plotting the resulting motion of this procedure the same way as the *Pocket* program in Figures 4.2 and 4.3, the diagrams should appear as a rectangle when looking at it from above and three step downs along a straight line when looking from the side.

When plotting the data logged from the *RapidBox* tool path it could be seen that the cutting corner behavior appeared also in these diagrams, as can be seen in Figure 4.4 and 4.5 with the robot speed of 25 mm/s. The speed of the robot affected the intensity of cutting corners which is exhibited in Figure 4.6 and 4.7, where the speed was 10 mm/s and then almost disappeard at a speed of 5 mm/s in Figure 4.8 and 4.9.

By plotting the data as points instead of lines it could be seen that there was loss of data while it were being logged. This data loss can be seen in Figure 4.10, it shows that the data were sampled with such a high-frequency that the plotted points appeared as lines. These lines are separated with spaces where it is evident that data were periodically being lost. The cutting corner appearance was caused by lines drawn between spaces, which was exhibited by adding dashed lines between the points, being evident in the corners in Figure 4.10. Figure 4.11 shows a close up of one of the corners from Figure 4.10. The points that are frequently plotted are clearly visible as well as the parts where the data were lost, causing the cutting corners appearance illustrated by the dashed lines.

Divide Logging

The logging of data was done by storing the sampled data during execution of a robot program into a log file and simultaneously converting it into a txt file. This txt file was then plotted with Kst. The command used for logging is given below and was executed in a Linux terminal.

```
orca_log -t 600 -o /dev/stdout --host localhost --port 2000
```

```
| orca_log2matlab /dev/stdin > /tmp/$USER/x.txt
```

The periodic loss of data was caused by the simultaneous conversion of data into the txt file. It seems like the logging of data stopped for a period while the data conversion was done and was then continued, causing loss of data during that period. The speed of the robot was therefore affecting the intensity of cutting corners due to the distance that the robot moved during the period that data was lost. With a higher speed the longer distance the robot moved during that period, appearing as more intense cutting of corners.

By dividing the logging and conversion into two parts, where the first part was the uninterrupted logging of data during the whole execution, the issue with losing data was solved. The command for this logging is given below.

```
orca_log -t 600 -o /tmp/$USER/myfile_log --host localhost
--port 2000
```

Once the logging of data was finished the second part was to convert the log file into a txt-file. The command used for this is given below.

```
orca_log2matlab /tmp/$USER/myfile_log >
/tmp/$USER/myfile_log.txt
```

When plotting the data after having used the divided logging method no data were lost, which can be seen in Figure 4.12. Without periods of data being lost, the points appear as thick lines without spaces due to the high-frequency sampling. No more cutting corner behavior appeared after this.

4.2 Trajectory vs. Path Tracking

As described in Section 2.2, the ILC problem formulation is to find an input to a system such that the output follows a desired trajectory, given a reference trajectory and a system [Norrlöf, 2000]. This formulation is why the robot's performance when tracking geometric trajectories was evaluated.

A geometric trajectory is a path through space given as a function of time. From experiments that tested the robot's performance when tracking trajectories, its ability to move to certain positions at given times was evaluated. It evaluates both the position accuracy as well as the synchronization between the robot's movements and reference. Errors between the reference and measured positions could therefore be caused by either timing issues, position errors or a combination of both.

Path tracking does not consider the time aspect and therefore it was tested when only evaluating the robot's position accuracy. In this project it was important that the accuracy of the robot was within the given maximum position error. It was not important whether a certain position was reached at a certain time. That is why the robot's performance when tracking a path was of interest. However, if the robot was

able to track trajectories with good results, the position error at a given time could be calculated. Trajectories makes implementation of ILC easier since the algorithms are given as functions of time, which is further described in Section 5. The position error could then be added to the original reference at a certain time as an input for the controller, which is why the robot's ability of trajectory tracking was tested.

Evaluation of Trajectory Tracking

The data logged from executions were plotted over time (samples) to evaluate the robot's performance when tracking trajectories. From diagrams that show the reference, measured position, and the error between them as position error per sample (*Point Index*) a large error could be seen. The errors varied between 1.5 mm to 4 mm, which is shown in Figures 4.13 - 4.15 and Figures 4.16 - 4.18. These errors were a lot larger than the specified robot accuracy of ± 0.5 mm [ABB Robotics, 2010], but since ExtCtrl was used the robot's accuracy was reduced. The feedforward of joint torques in the ABB IRC5 axes controllers were disconnected when ExtCtrl was used, which reduced the robot's accuracy, but the errors were still significantly larger than expected.

The error was calculated by first setting the start positions of the reference and measured positions to zero before running the program. It was then calculated as relative movement between the reference and measured position in relation to the *base* frame.

When looking at the diagrams over areas where the robot was making moves in x, y and z directions, it could be seen that there was a difference in samples between reference and measured positions. This sample difference can be seen in Figures 4.19 - 4.21 where it is evident that the moves in reference positions start and end on different times compared to the measured positions. Figures 4.22 - 4.24 illustrate the sample difference of the logged data with vertical lines showing where the start and end of the moves are.

The exact cause of the different start and end times that appear in the diagrams were not determined. They could be caused by a delay that occurred when the signals were processed, mechanical lag or by a combination of both. ORCA handled all the signals that were logged from the model made in Simulink and it was when analyzing this data that the delay of samples could be seen, but it was not determined where it occurred.

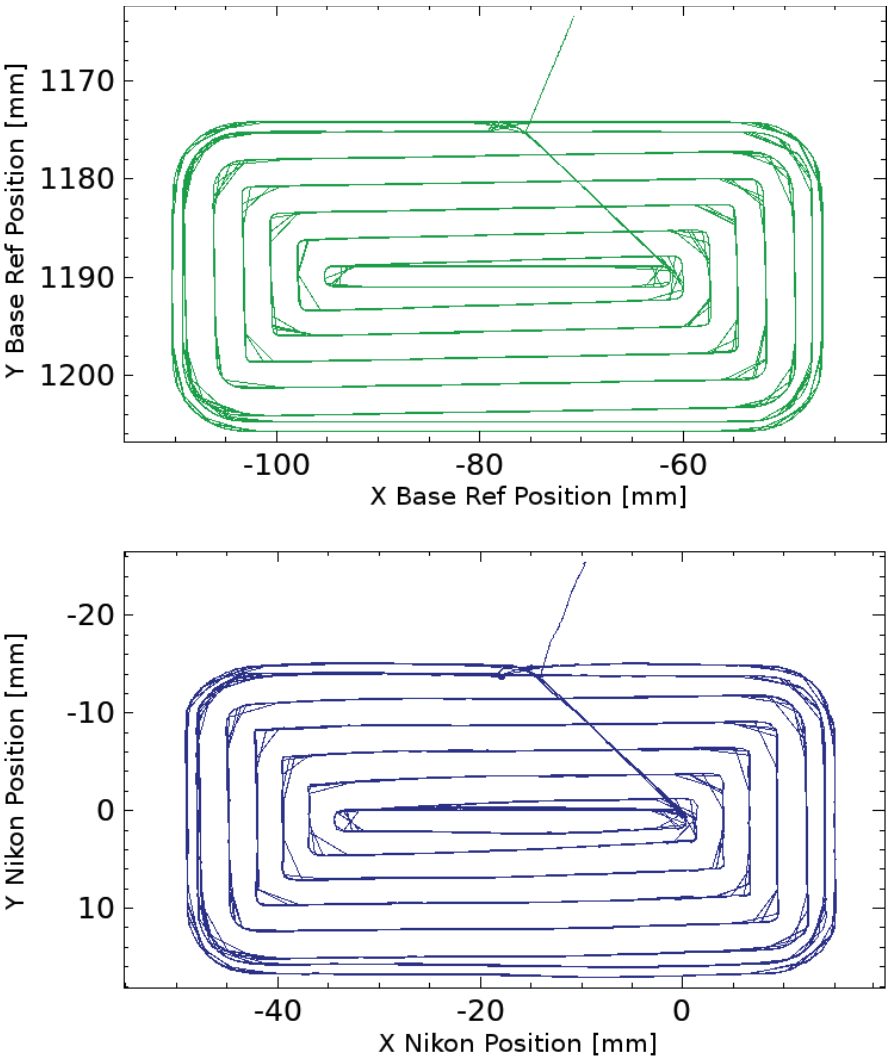


Figure 4.2 Diagrams logged from the *Pocket* tool path executed with a velocity of 50 mm/s, showing the reference (top plots) and the measured positions (bottom plots) relative the *base* frame.

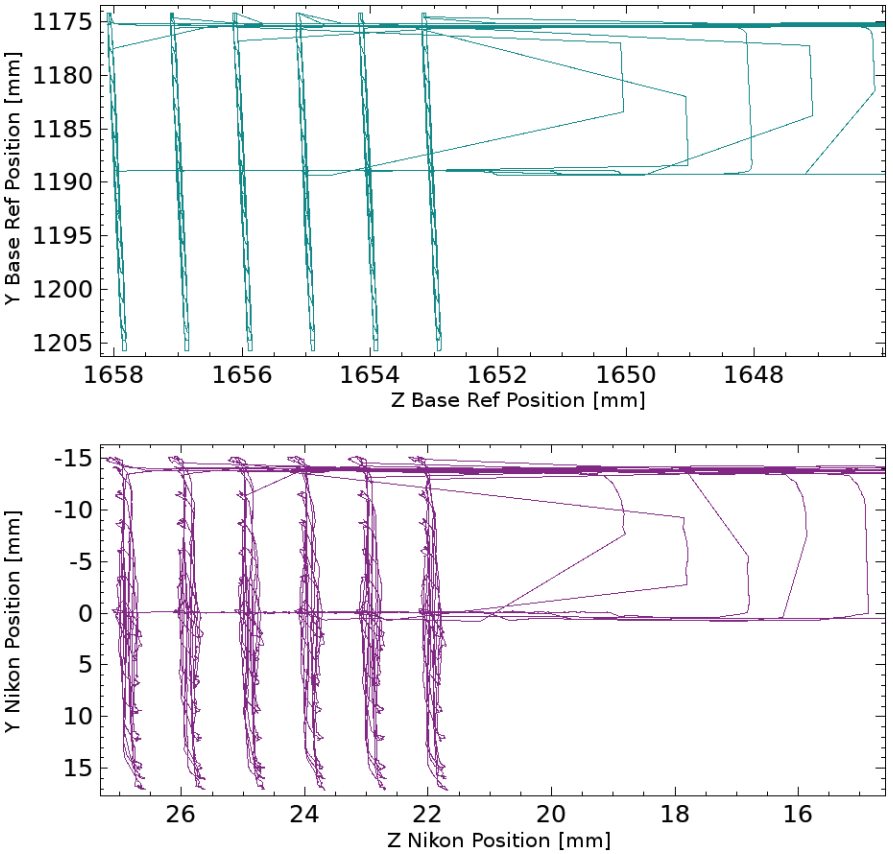


Figure 4.3 Diagrams logged from the *Pocket* tool path executed with a velocity of 50 mm/s, showing the reference (top plots) and the measured positions (bottom plots) relative the *base* frame.

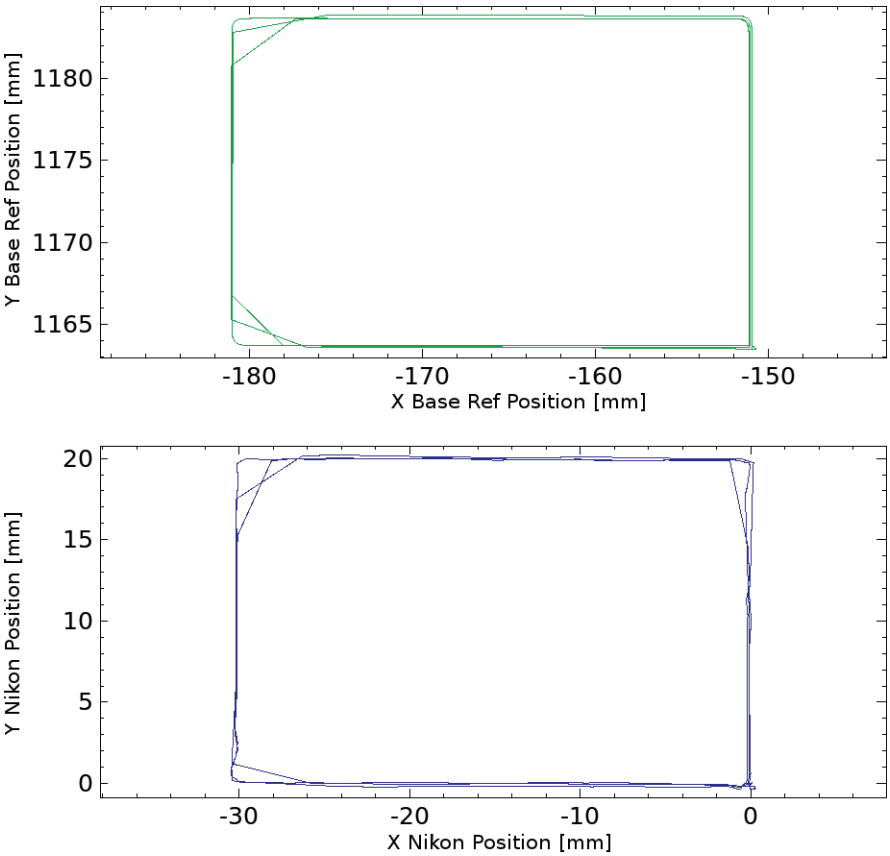


Figure 4.4 Diagrams from the *RapidBox* tool path executed with a velocity of 25 mm/s with the reference and the measured positions.

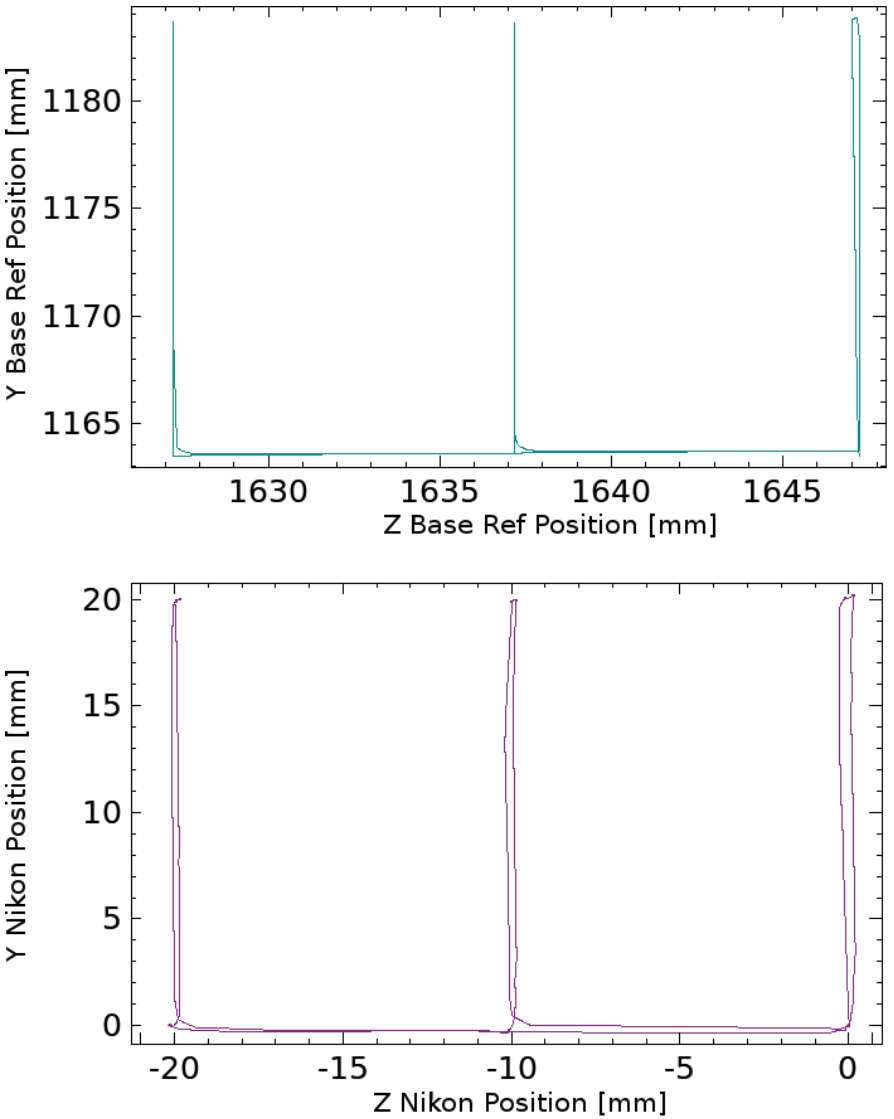


Figure 4.5 Diagrams from the *RapidBox* tool path executed with a velocity of 25 mm/s with the reference and the measured positions.

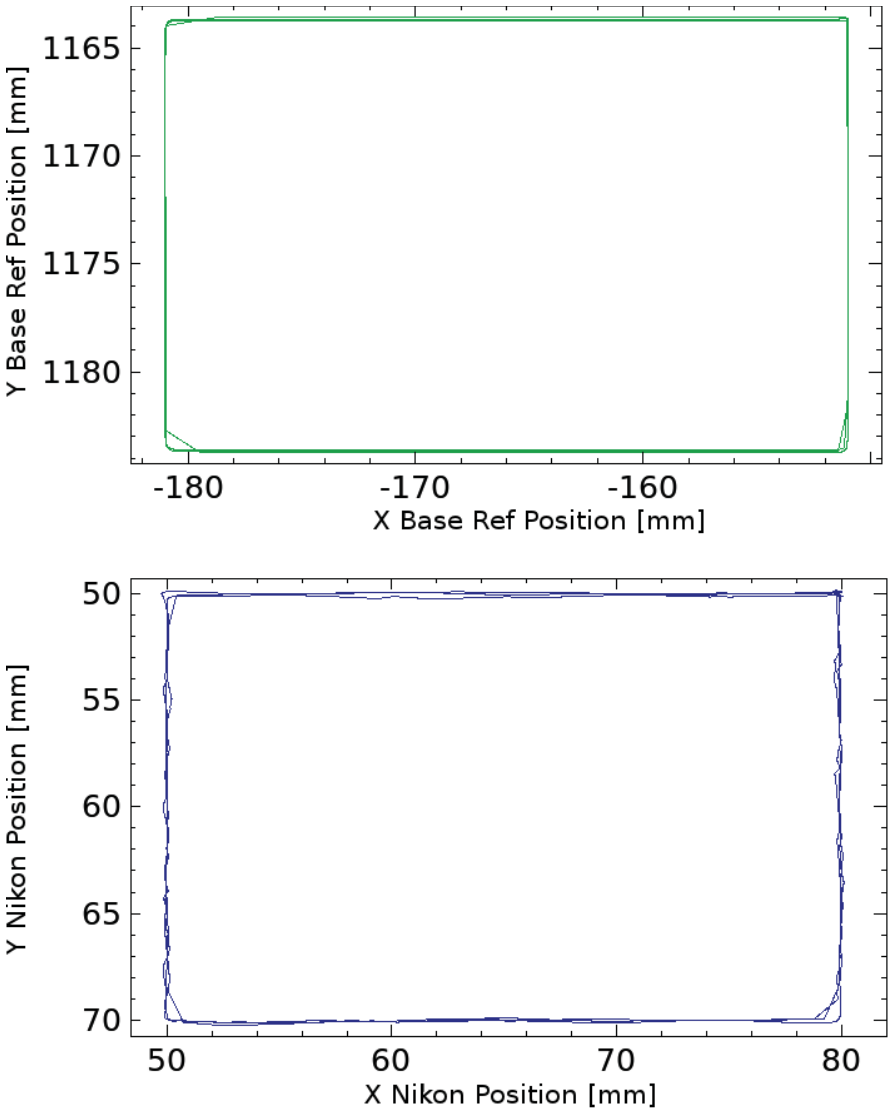


Figure 4.6 Diagrams from the *RapidBox* tool path executed with a velocity of 10 mm/s with the reference and the measured positions.

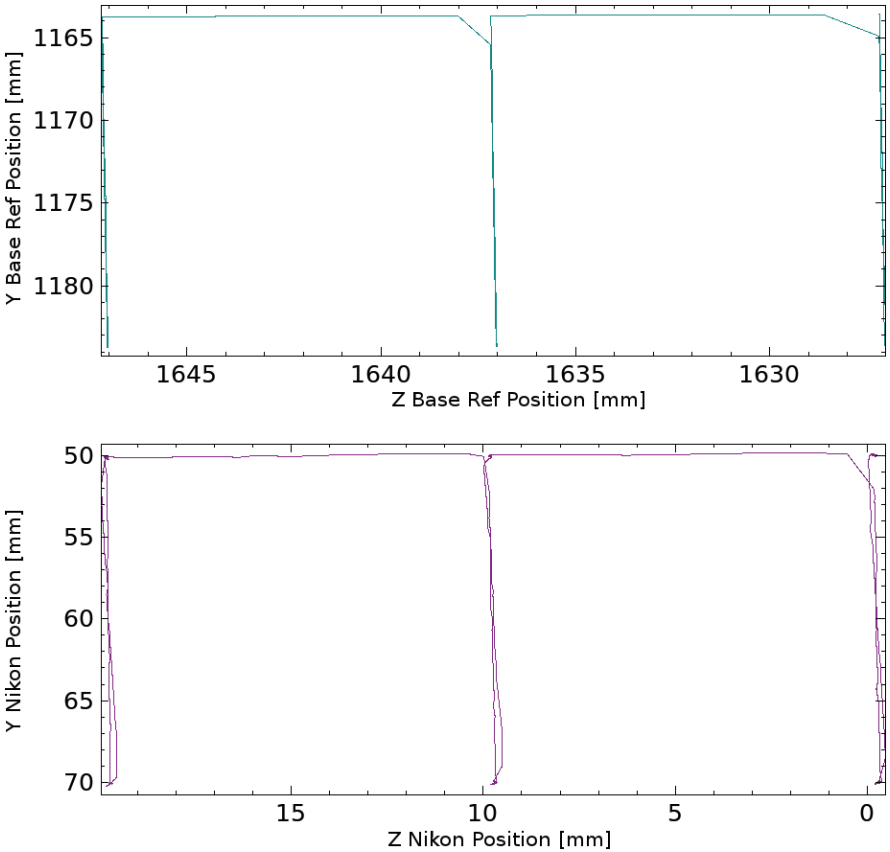


Figure 4.7 Diagrams from the *RapidBox* tool path executed with a velocity of 10 mm/s with the reference and the measured positions.

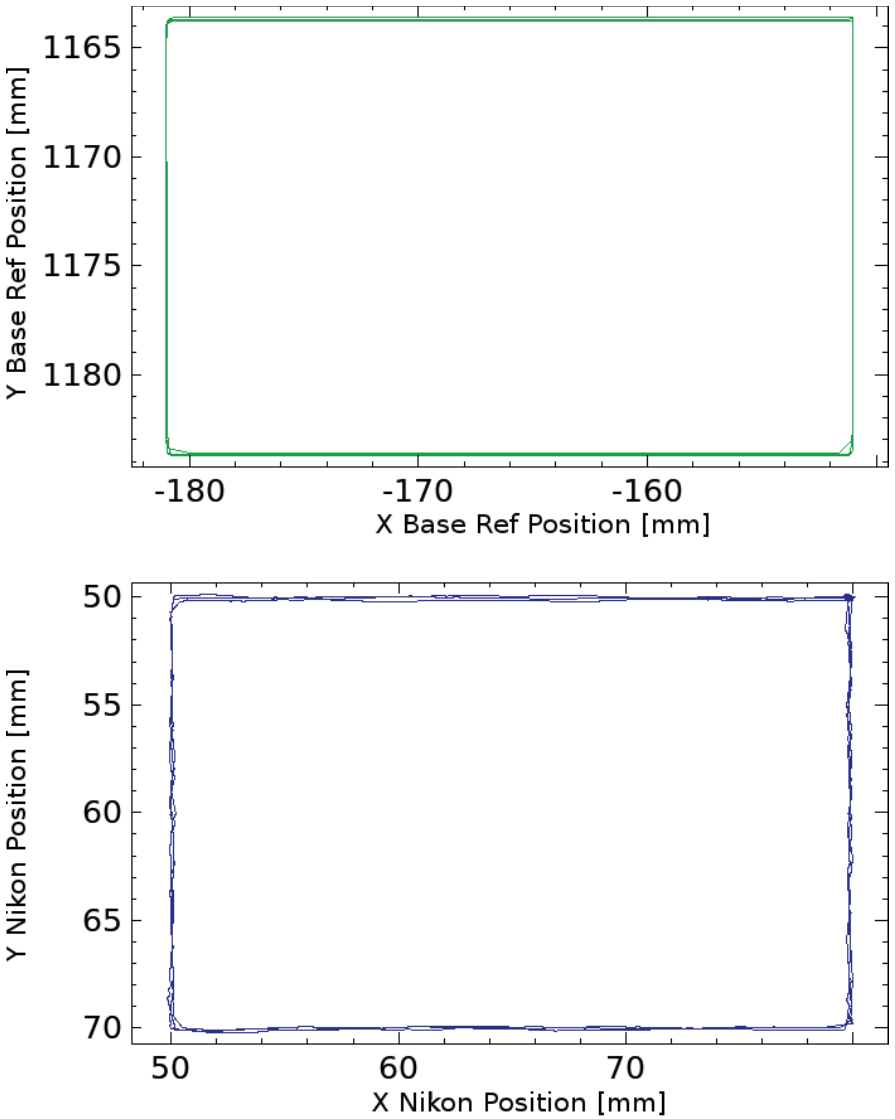


Figure 4.8 Diagrams from the *RapidBox* tool path executed with a velocity of 5 mm/s with the reference and the measured positions.

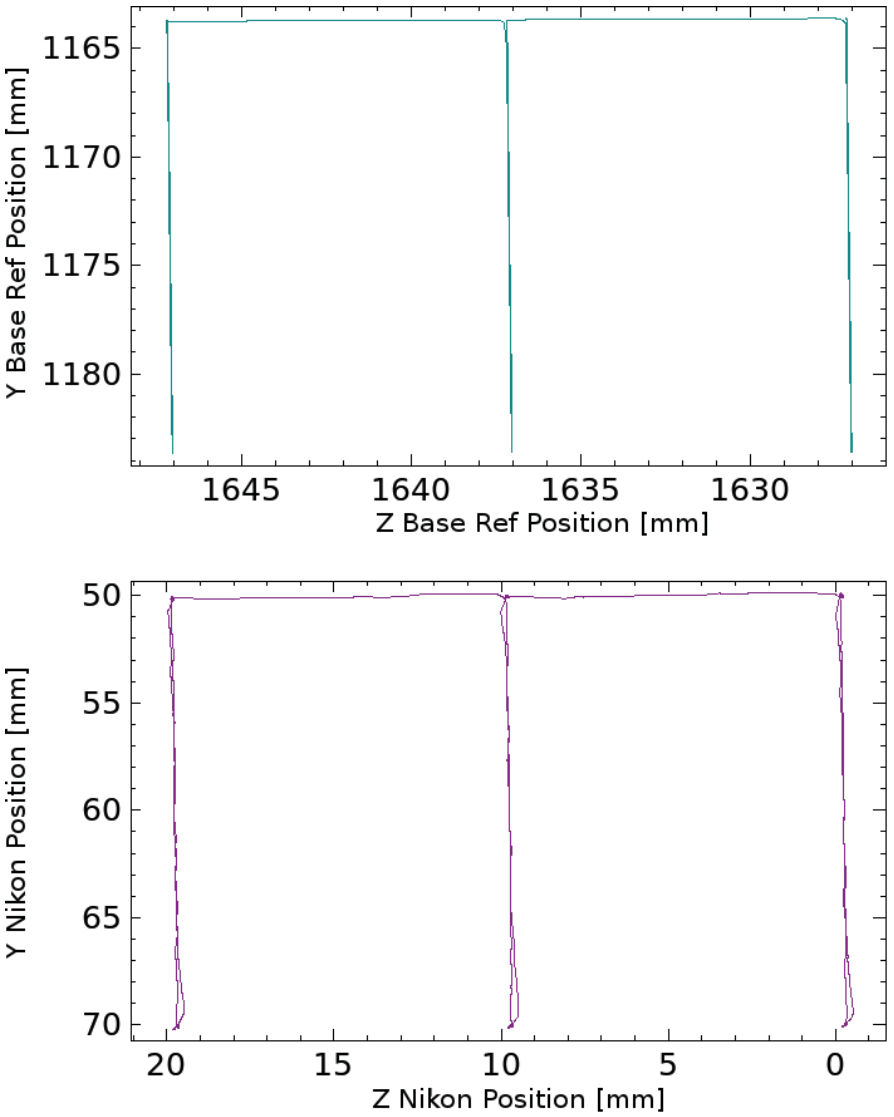


Figure 4.9 Diagrams from the *RapidBox* tool path executed with a velocity of 5 mm/s with the reference and the measured positions.

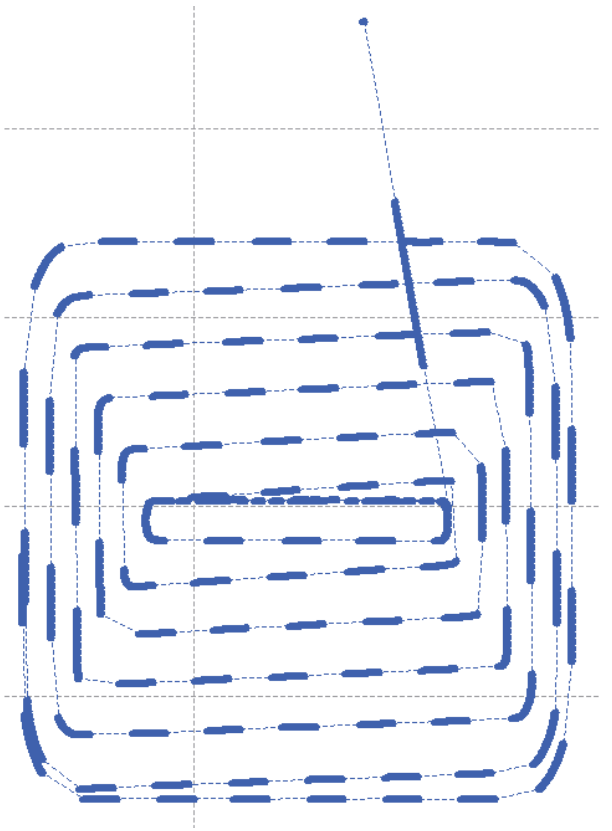


Figure 4.10 Data plotted as points, appearing as thick lines because of the high-frequency sample of data. Between the points are dashed lines that are visible over the spaces where data was lost.

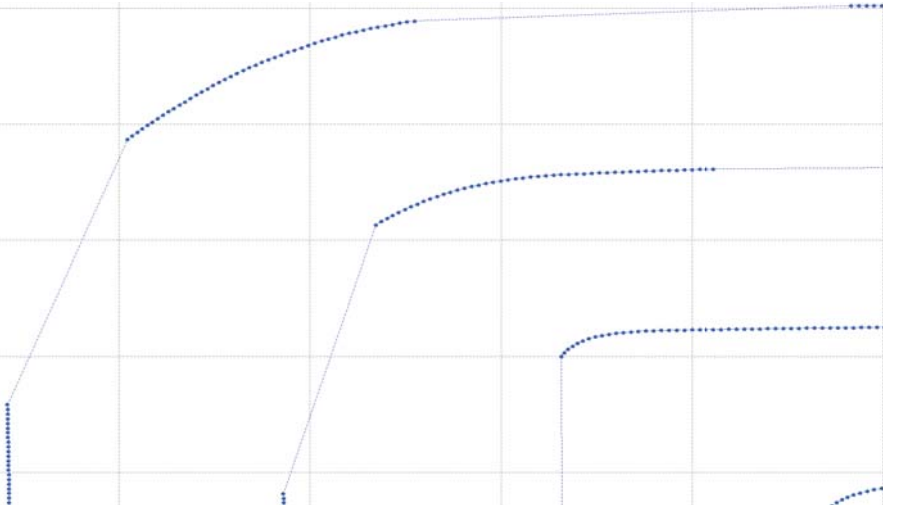


Figure 4.11 Close up on a corner with clear visibility of the points and the spaces between them, connected with dashed lines. The lines between spaces causes the cutting corner appearance.

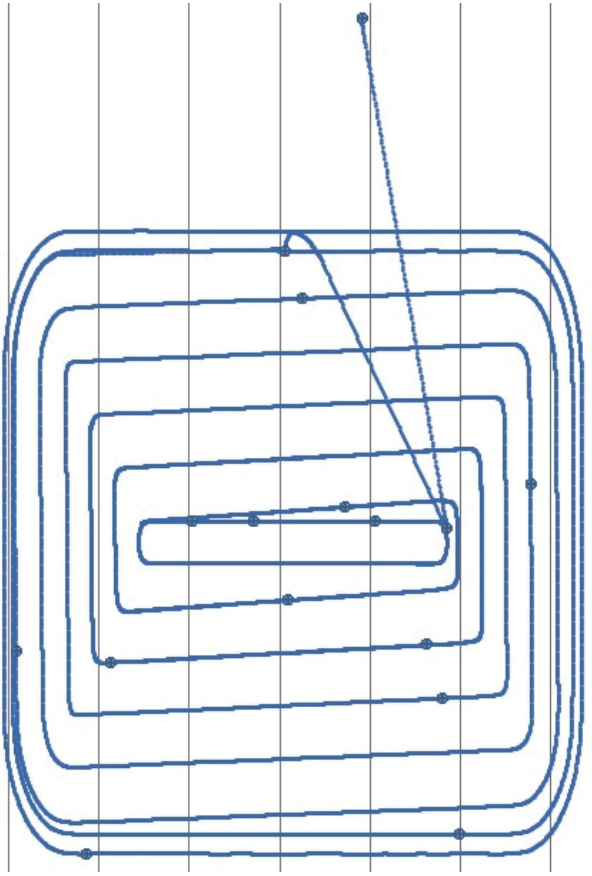


Figure 4.12 Plot of data as points after using the divided logging method.

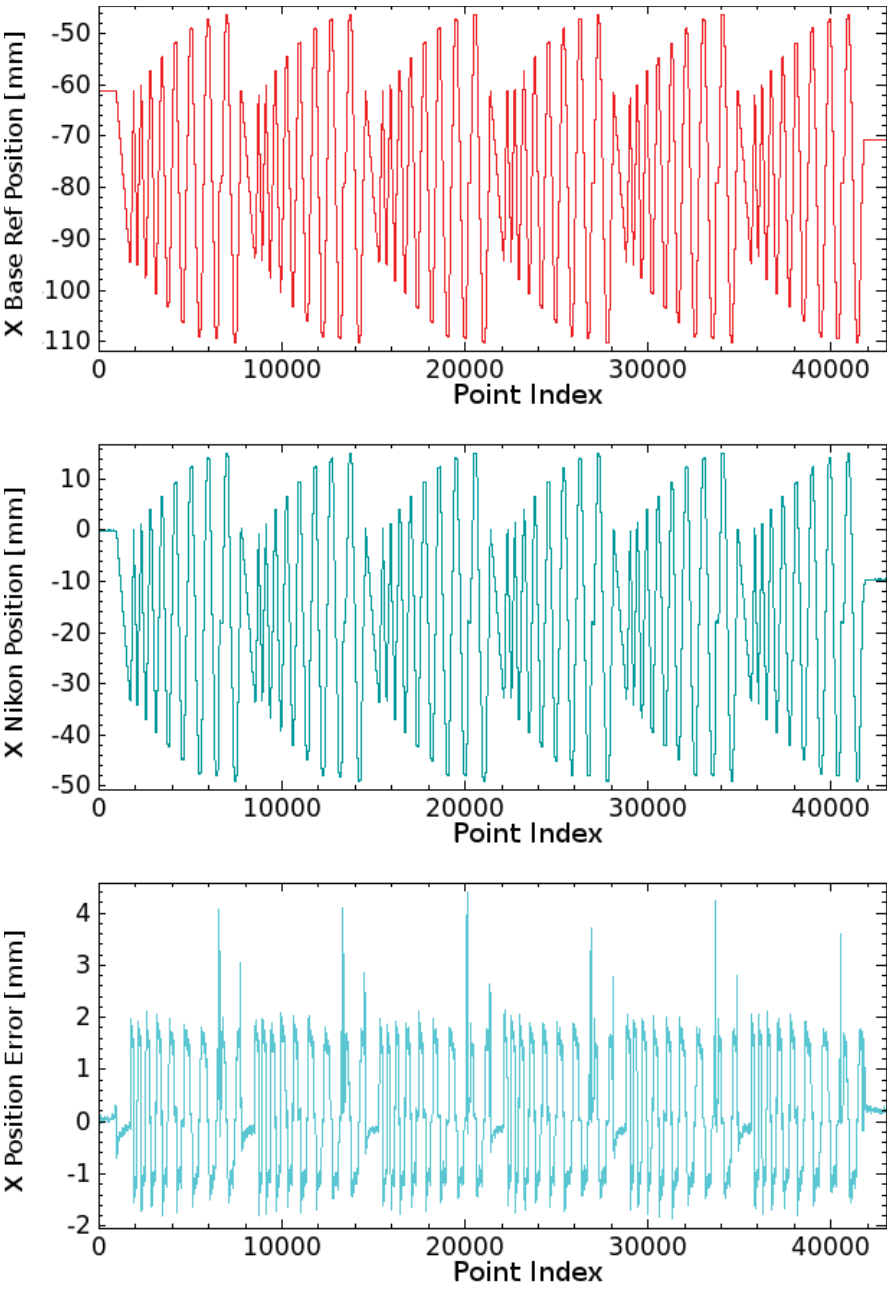


Figure 4.13 Diagrams of *Pocket* tool path with reference, measured positions, position error and, samples (*Point Index*) [250 Hz].

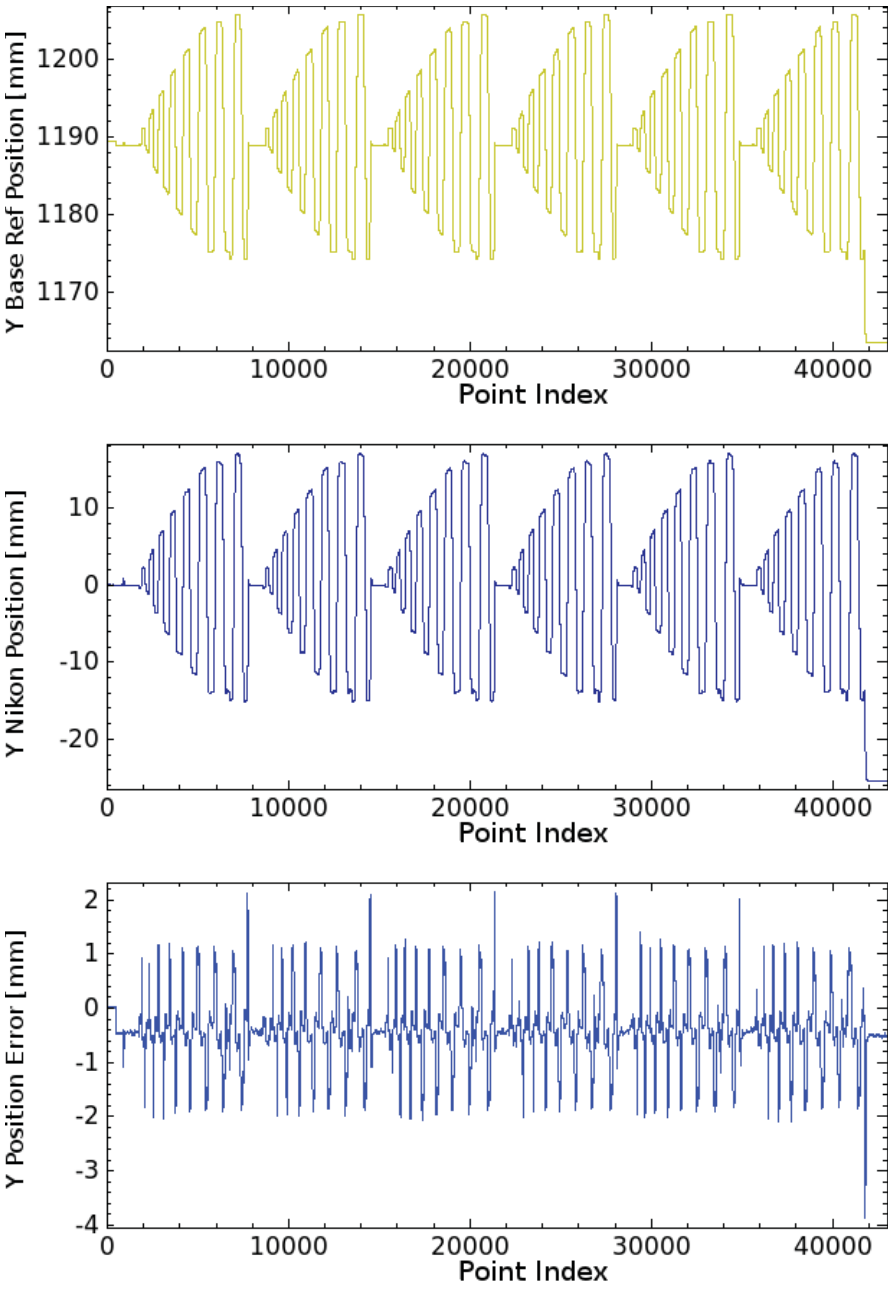


Figure 4.14 Diagrams of *Pocket* tool path with reference, measured positions, position error and, samples (*Point Index*) [250 Hz].

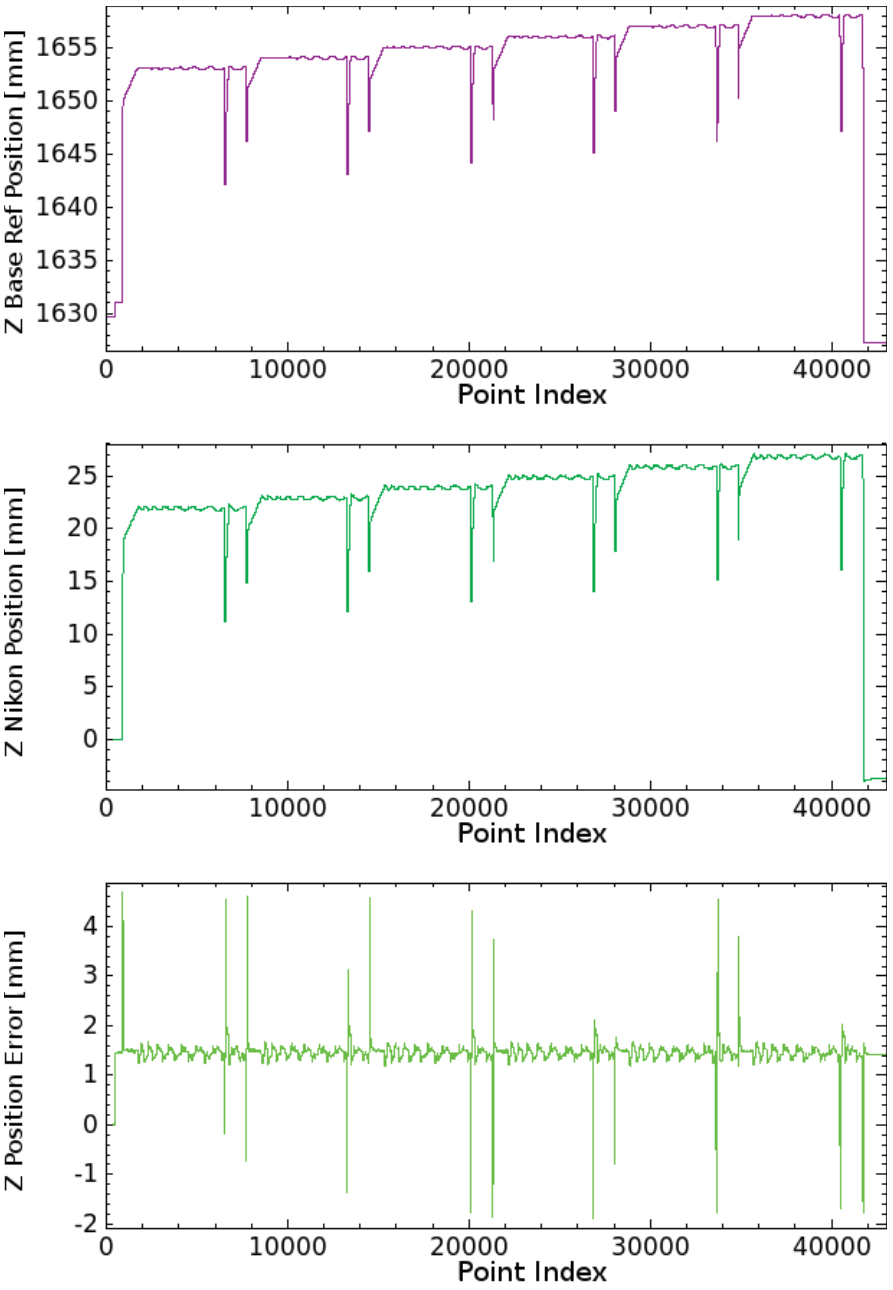


Figure 4.15 Diagrams of *Pocket* tool path with reference, measured positions, position error and, samples (*Point Index*) [250 Hz].

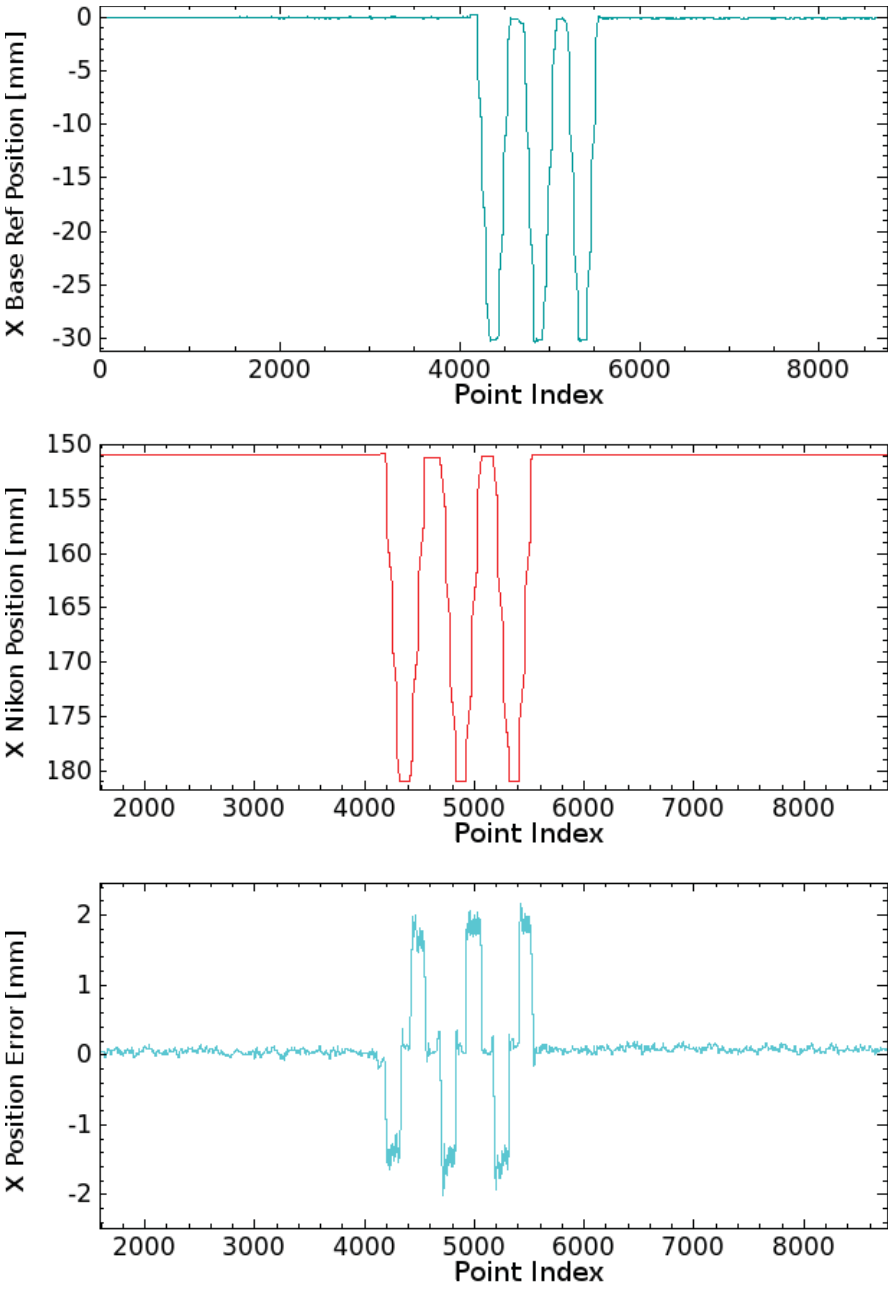


Figure 4.16 Diagrams of *RapidBox* tool path with reference, measured positions, position error and, samples (*Point Index*) [250 Hz].

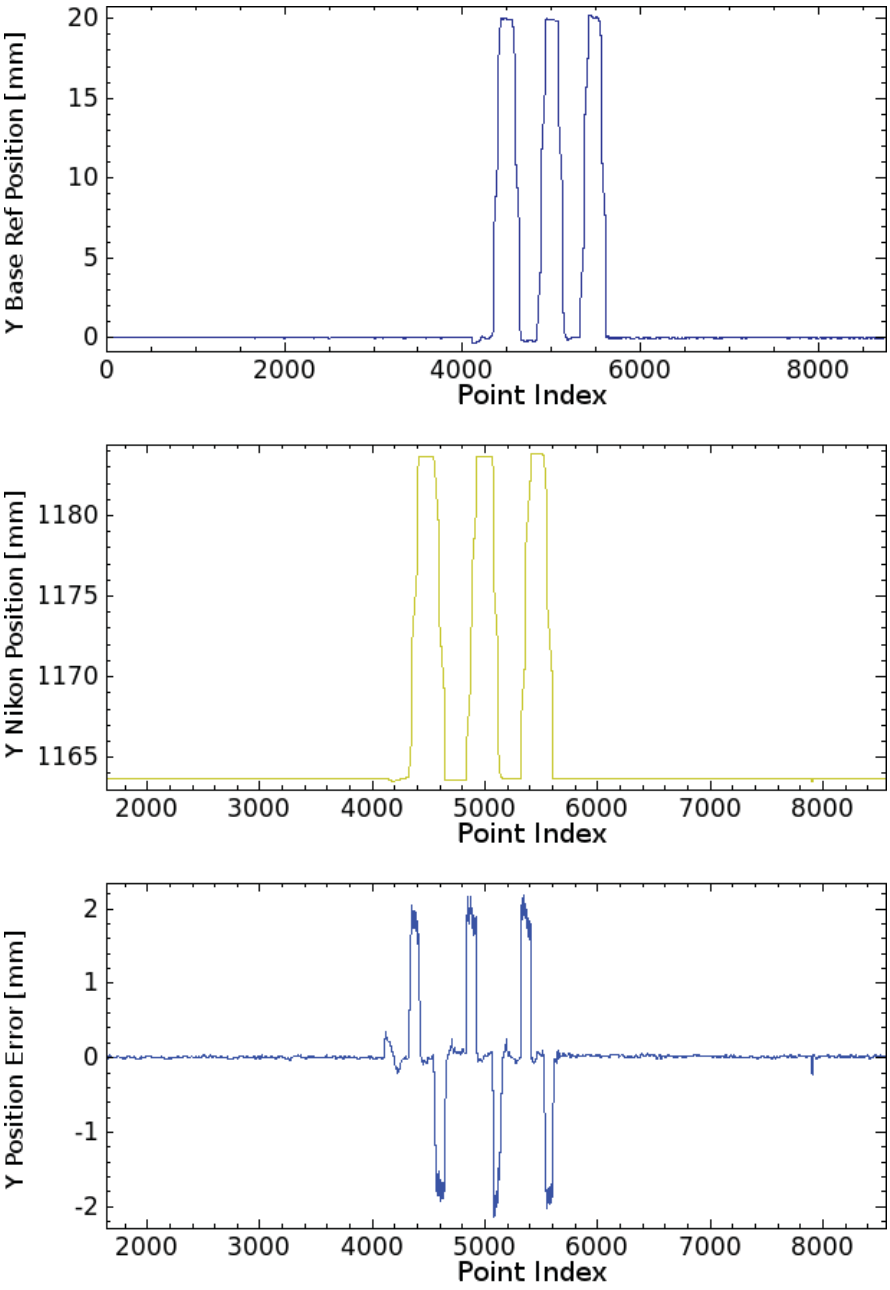


Figure 4.17 Diagrams of *RapidBox* tool path with reference, measured positions, position error and, samples (*Point Index*) [250 Hz].

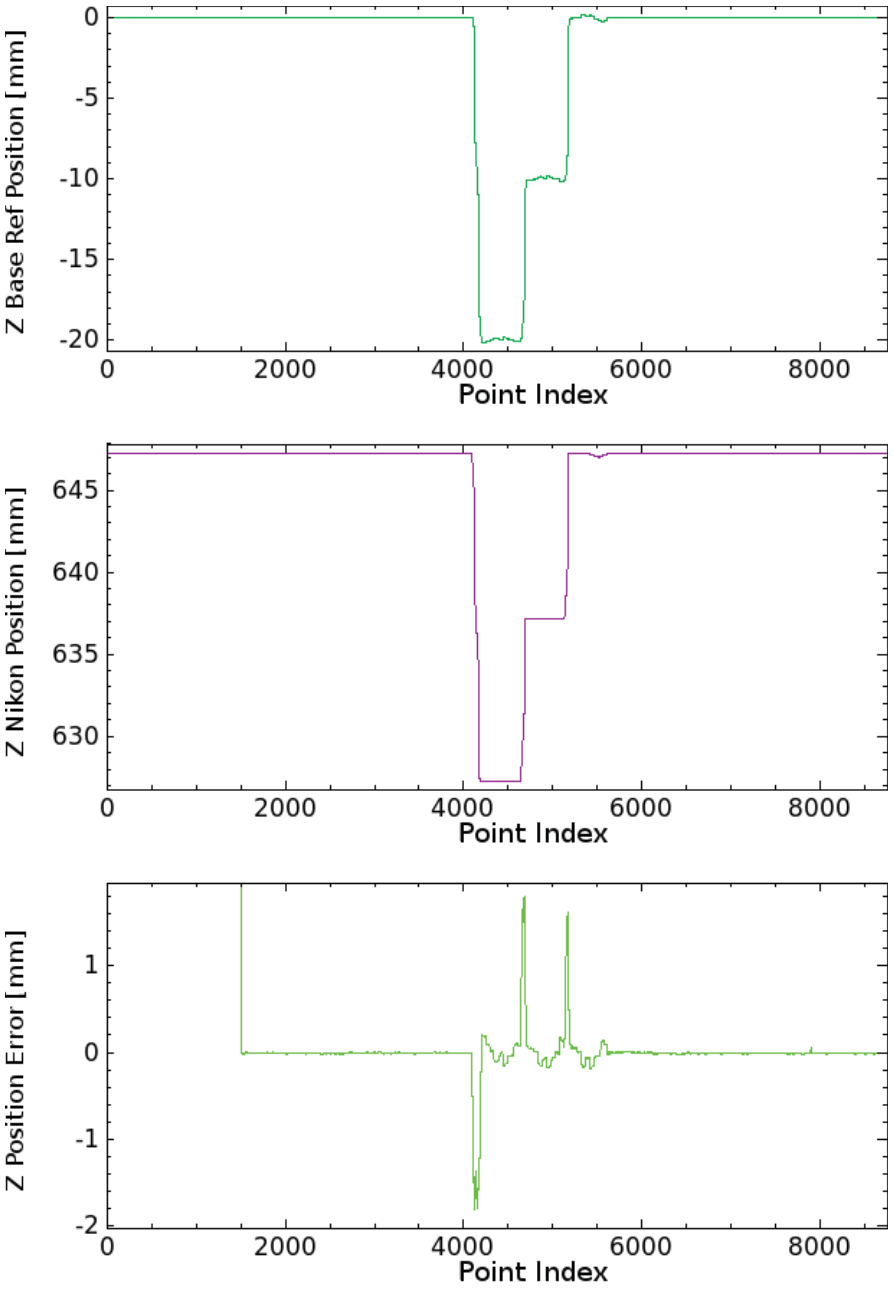


Figure 4.18 Diagrams of *RapidBox* tool path with reference, measured positions, position error and, samples (*Point Index*) [250 Hz].

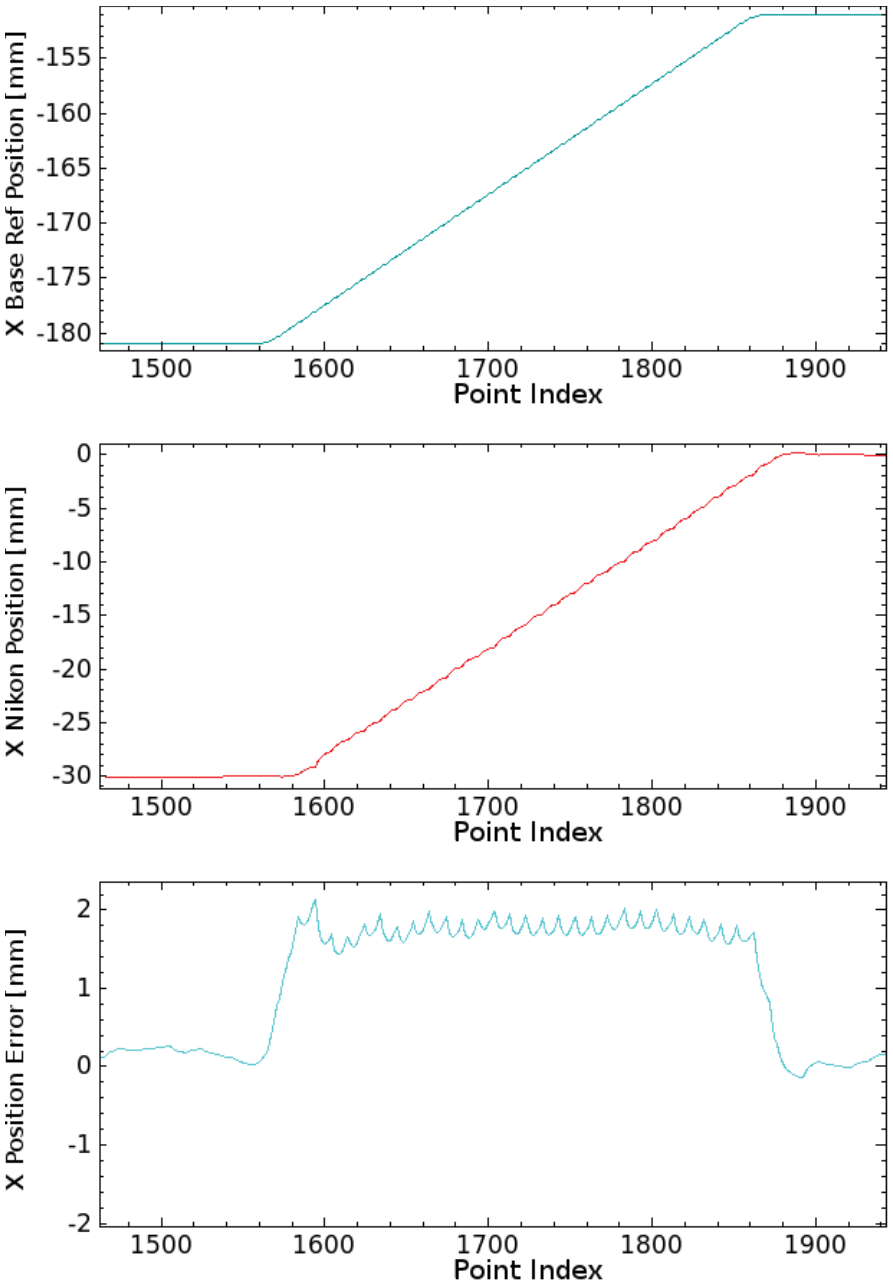


Figure 4.19 Zoom on movements (*RapidBox* tool path) with reference, measured positions, position error and, samples (*Point Index*) [250 Hz]).

When the robot moved it started by accelerating to the specified robot speed and would then decelerate until it stopped at the desired position. Since the acceleration was registered in the reference first, the error would increase until the move started in the measured position, which eventually resulted in a constant error. The size of the constant error depends on the speed of the robot. At the end of the move the deceleration reference would decrease the error. When both the reference and the position had completed the move, the error went down to vary around zero again. This appearance gave a step-shaped appearance when plotting the position error as seen in Figures 4.19 - 4.21 and Figures 4.22 - 4.24.

The step-shaped errors caused by the different start and end times illustrate the sample delay that appeared between the signals in the diagrams. Because of the delayed samples the position error was showing a time-based error and not the difference between the desired and actual robot position from following the desired path, which was the important error. This error could therefore not be used as an input for the controller without estimating and compensating for the delay.

Model Compensation

From the diagrams the sample difference was estimated for the tested trajectory and then added to the reference signal in the Simulink model. It was estimated to be approximately 18 samples for this trajectory and by testing different values the best result was given by delaying 18 samples. With the Simulink block *Integer Delay* the number of samples that should delay the reference signal could be added. An overview of the complete Simulink model is given in Appendix A.2.

The results showed that by delaying the reference for this trajectory with the estimated samples, the measured positions were aligned and the calculated trajectory tracking error was significantly reduced. It was reduced from an average of 2 mm to about 0.3 mm, which was within the specified robot accuracy of ± 0.5 mm. Figures 4.25 - 4.27 show the result of the sample delay compensation with the signals aligned and the error varying between 0.2 and 0.4 mm. This compensation was, however, not a generic solution since the amount of samples to be delayed must be specified for each trajectory.

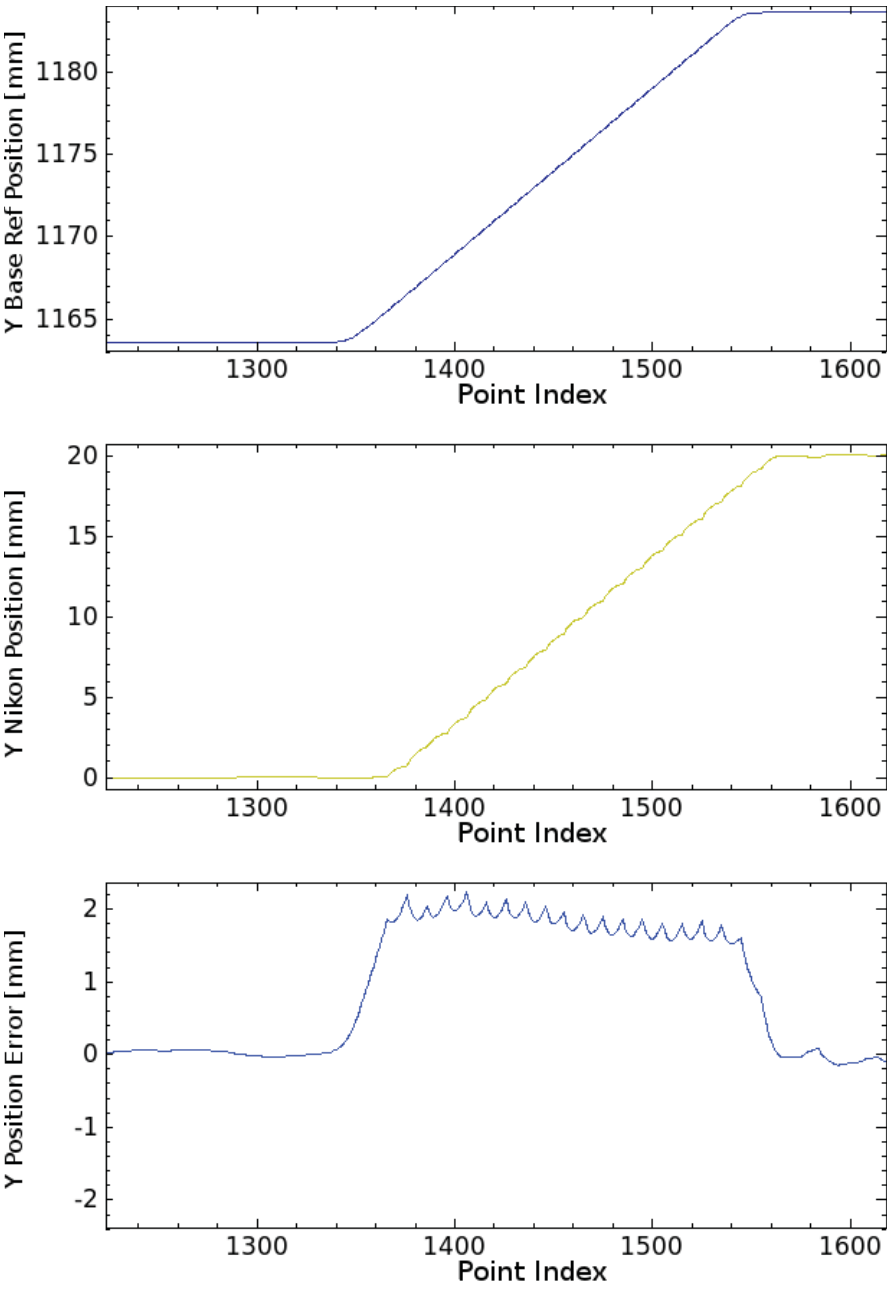


Figure 4.20 Zoom on movements (*RapidBox* tool path) with reference, measured positions, position error and, samples (*Point Index*) [250 Hz]).

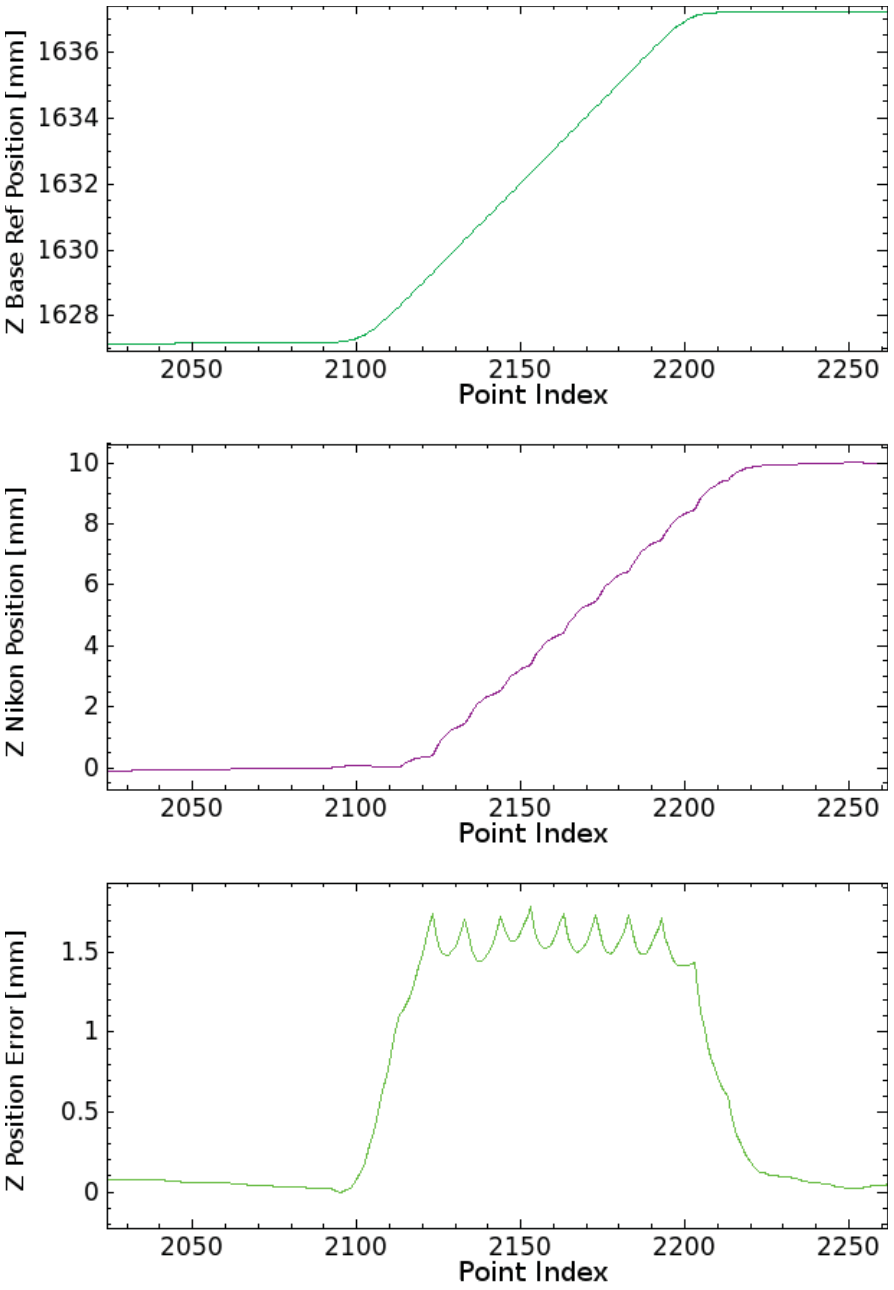


Figure 4.21 Zoom on movements (*RapidBox* tool path) with reference, measured positions, position error and, samples (*Point Index*) [250 Hz]).

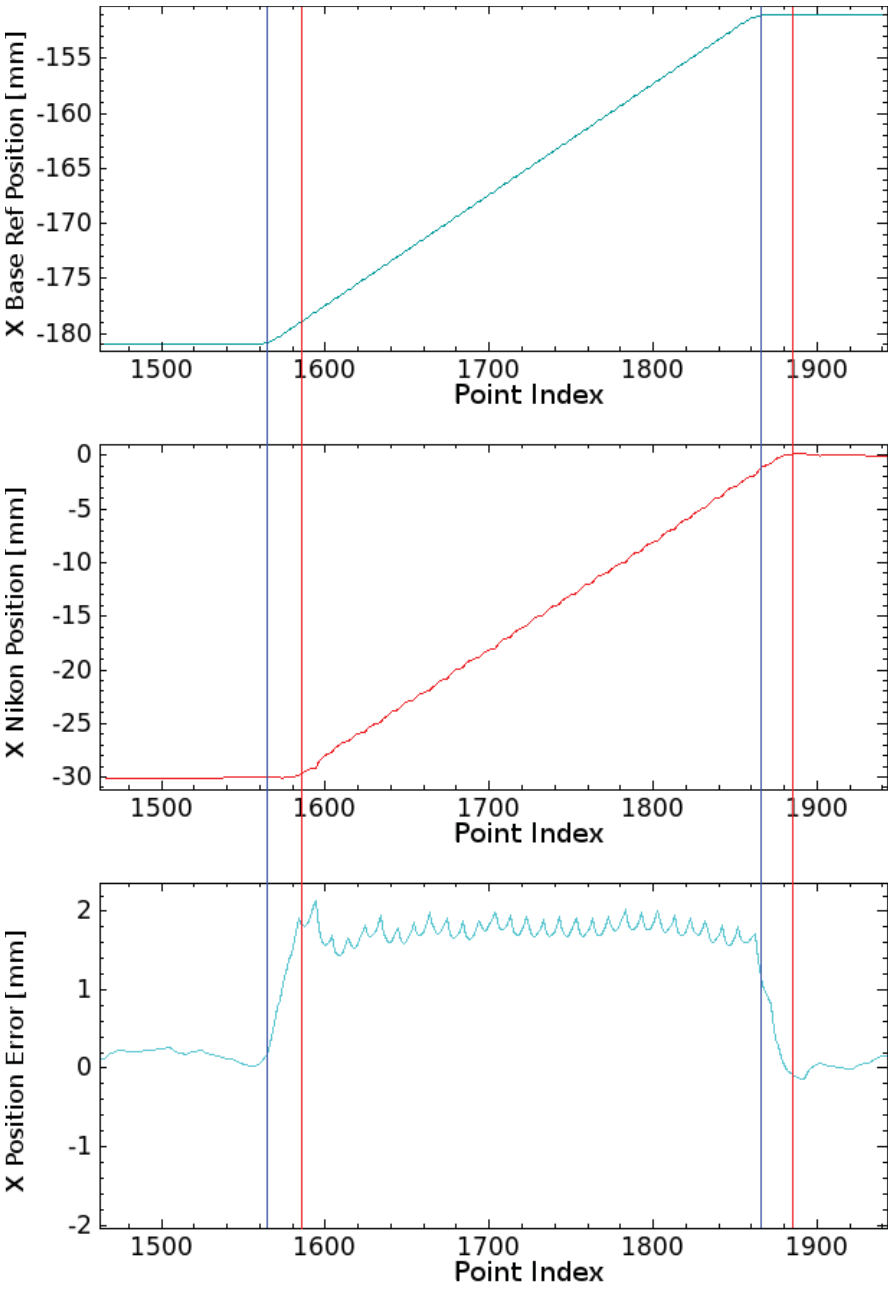


Figure 4.22 Zoom on movements (*RapidBox* tool path) with vertical lines showing start and end samples of reference (blue) and measured positions (red) and the position error and, samples (*Point Index*) [250 Hz].

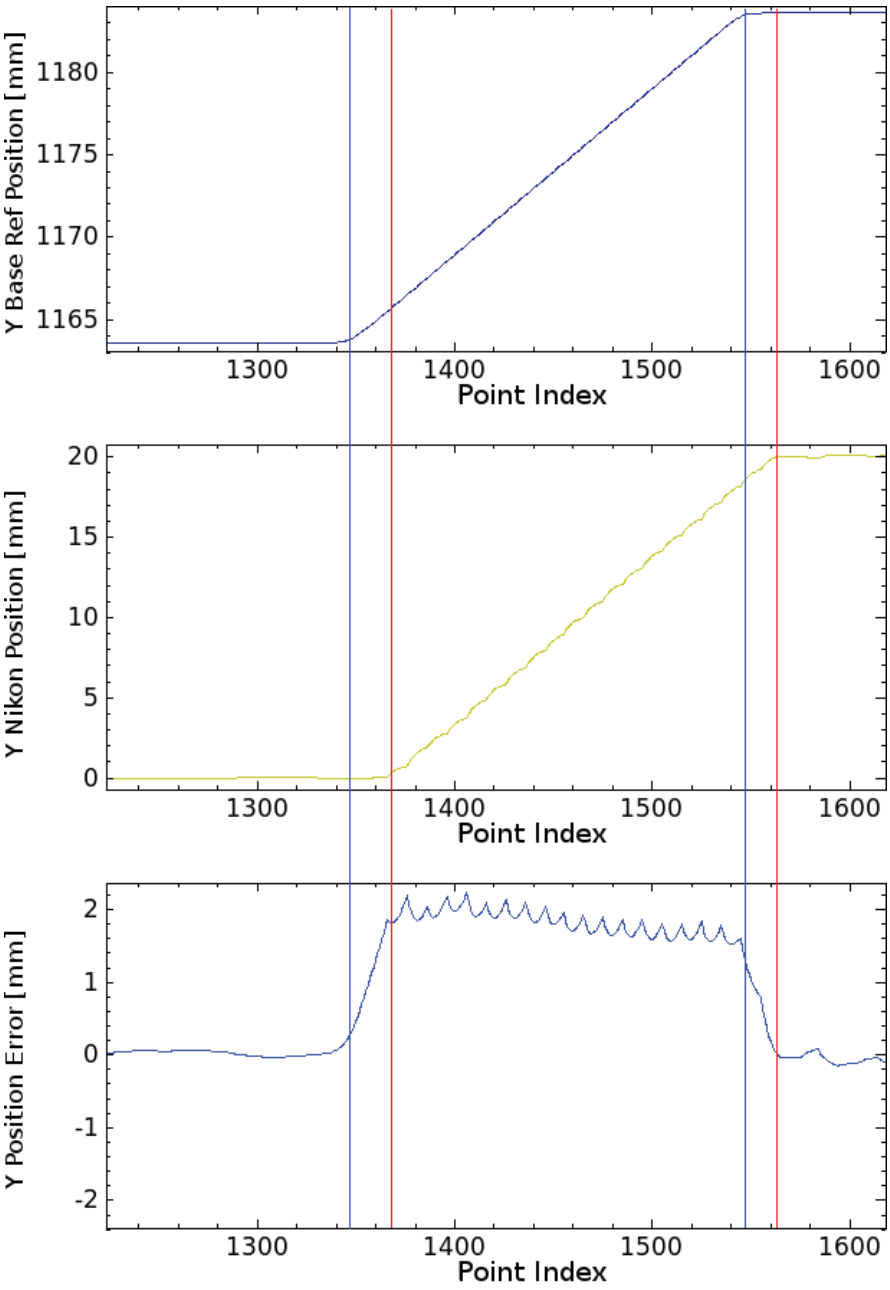


Figure 4.23 Zoom on movements (*RapidBox* tool path) with vertical lines showing start and end samples of reference (blue) and measured positions (red) and the position error and, samples (*Point Index*) [250 Hz].

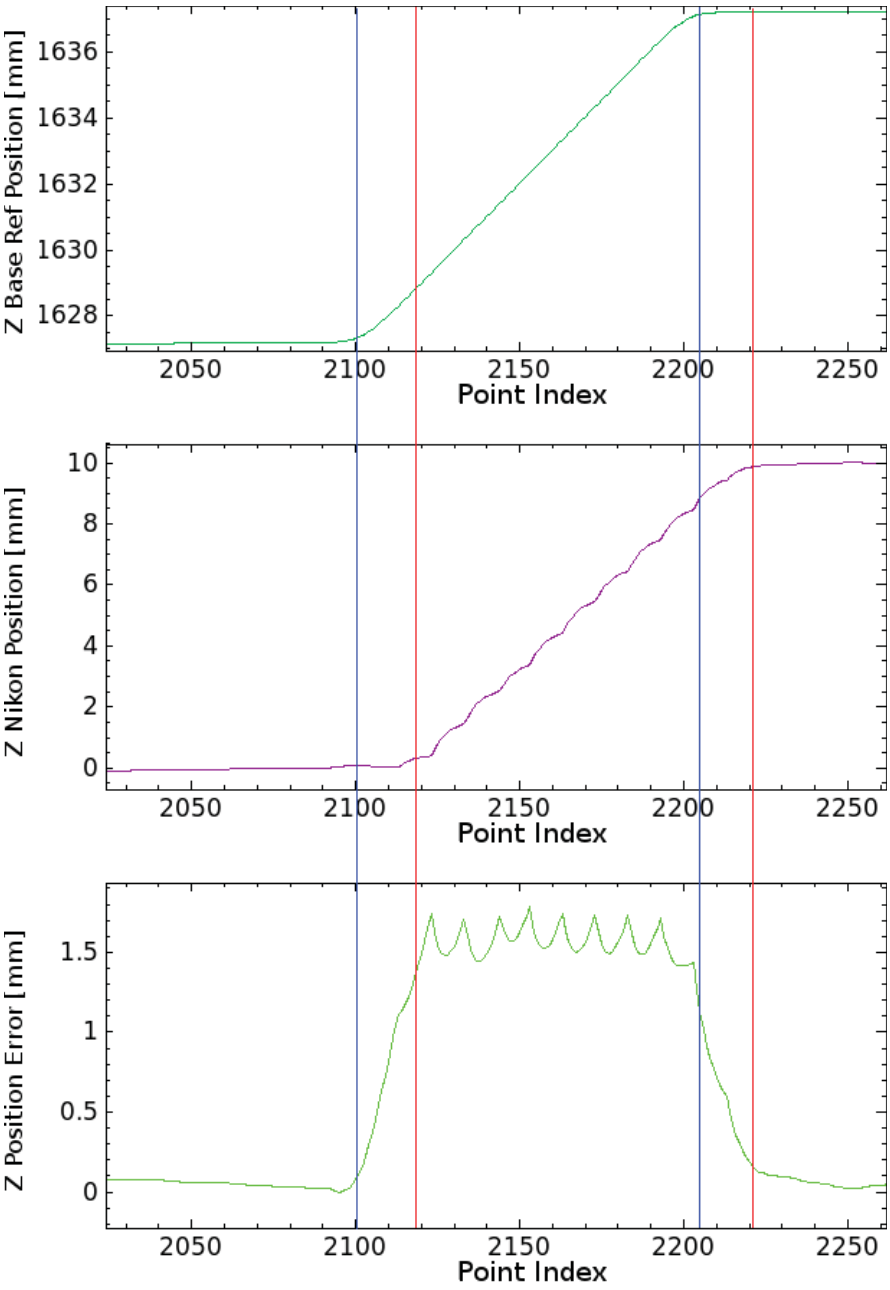


Figure 4.24 Zoom on movements (*RapidBox* tool path) with vertical lines showing start and end samples of reference (blue) and measured positions (red) and the position error and, samples (*Point Index*) [250 Hz].

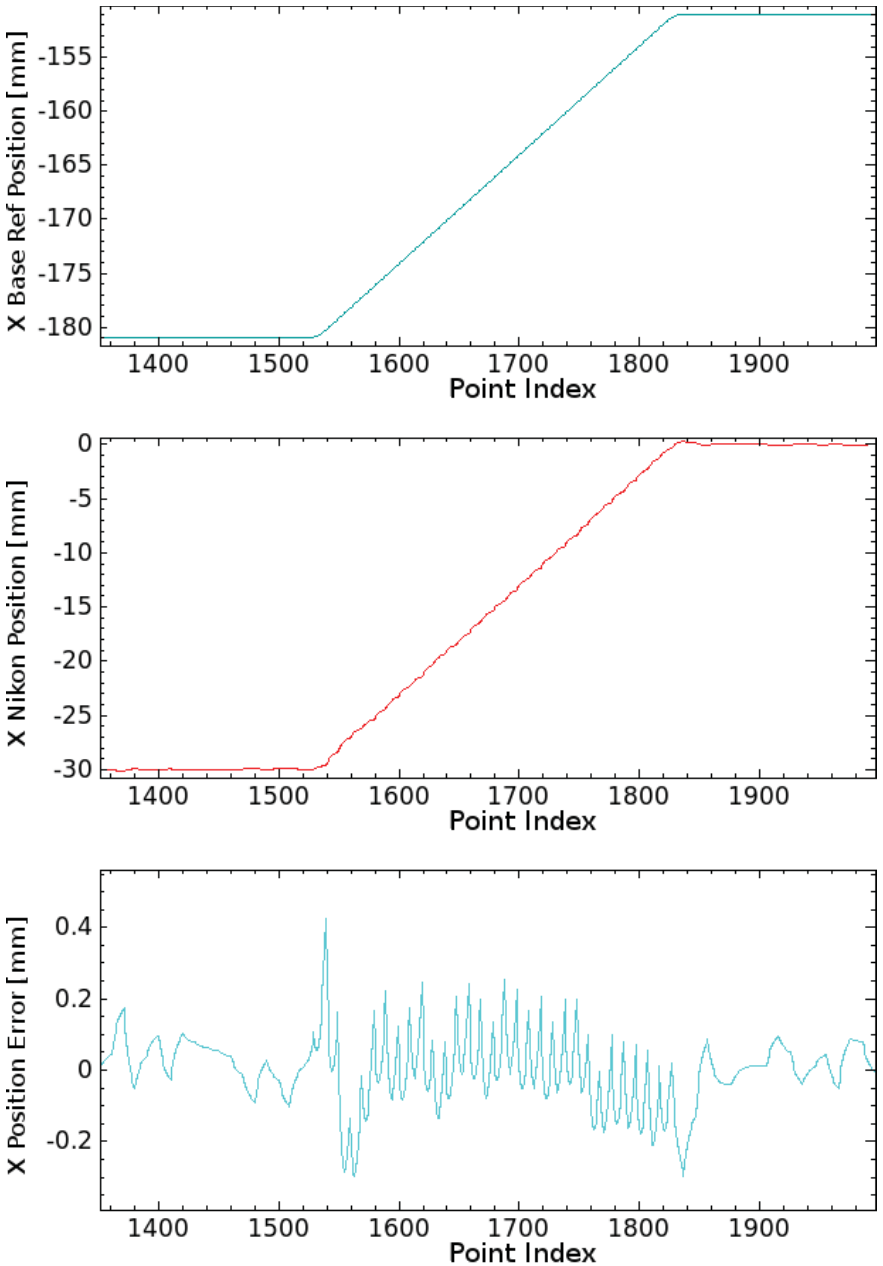


Figure 4.25 Results from delay compensation of 18 samples [250 Hz]. The reference positions and measured positions are aligned.

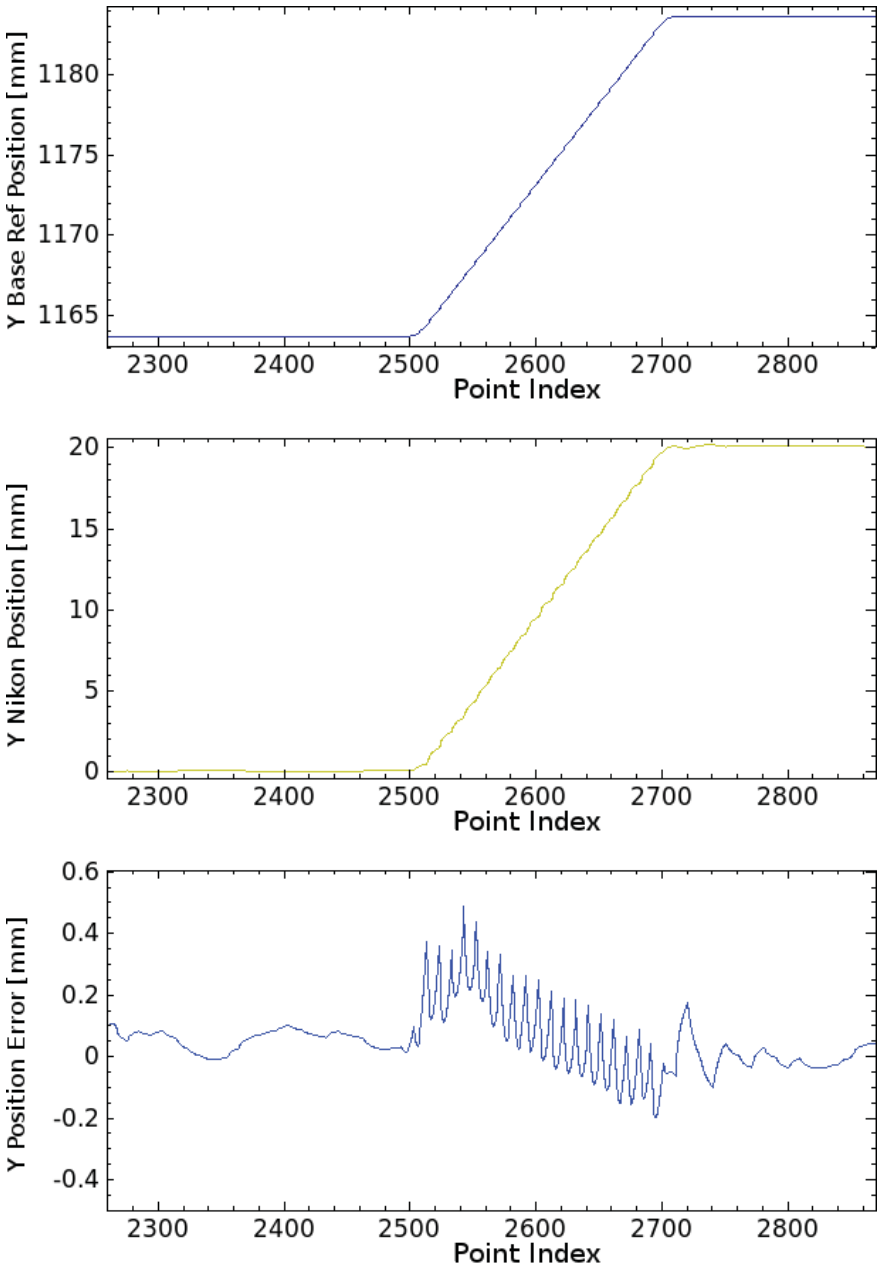


Figure 4.26 Results from delay compensation of 18 samples [250 Hz]. The reference positions and measured positions are aligned.

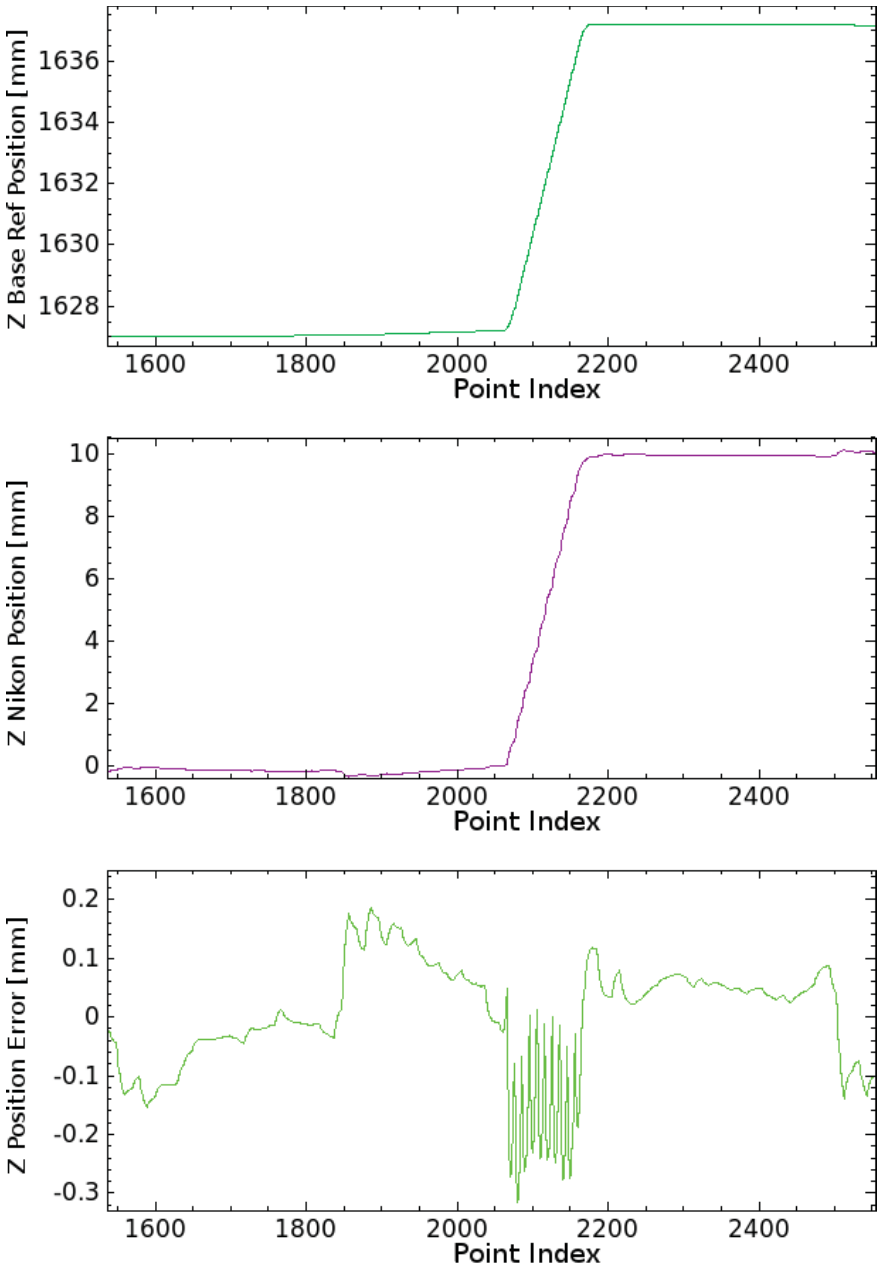


Figure 4.27 Results from delay compensation of 18 samples [250 Hz]. The reference positions and measured positions are aligned.

5

Method

This chapter describes the methods used to implement ILC algorithms in milling to improve the robot's accuracy. It describes the script that was used to create the reference tool paths, that were executed by the robot and how the measured data from these executions were used to generate ILC tool paths.

5.1 ILC Script

Matlab was used to create a script that could calculate and generate ILC compensated tool paths in three geometric dimensions. The script could read and analyze chosen data log files and calculate compensated tool paths using ILC algorithms. It could then translate the new tool path into RAPID code that could be sent to the robot.

Reference Deviation

Before the robot received a cartesian position reference it was processed in the IRC5 robot controller to make the robot move along the desired positions as accurately as possible. Model-based dynamic deviations that could occur at different robot positions were compensated for by the feed-forward in the controller. This IRC5 robot controller compensation made the reference signal deviate from the original reference, which was used for calculating the error and could therefore not be used for making ILC compensated tool paths. It would cause the ILC compensated reference to contain both the deviations generated from the robot controller and the calculated error. When sending this reference to the robot it would then be changed once more within the robot controller. That is why all measured positions must be compared to an original reference without adjustments.

In Figure 5.1 the nominal *RapidBox* tool path reference (red), described in Section 4.1, was compared to the processed reference (blue). The references should both have appeared as simple rectangles when x was plotted versus y. However, the reference sent to the robot appeared with deviations when compared to the nominal tool path. A zoom of the corner shows this deviation.

This reference deviation was caused by the ABB IRC5 robot controller's compensation for dynamic position deviations when interpreting RAPID programs. The compensation was done before the reference was logged with ExtCtrl and therefore it could not be used for calculating ILC tool paths since the sensor data was compared to the processed reference. To calculate ILC tool paths the measured data (sensor data) must be compared to the original reference without the compensation. Figure 5.2 illustrates how the RAPID program was processed by the ABB IRC5 robot controller before it was logged and monitored through ExtCtrl to produce the diagram seen in Figure 5.1.

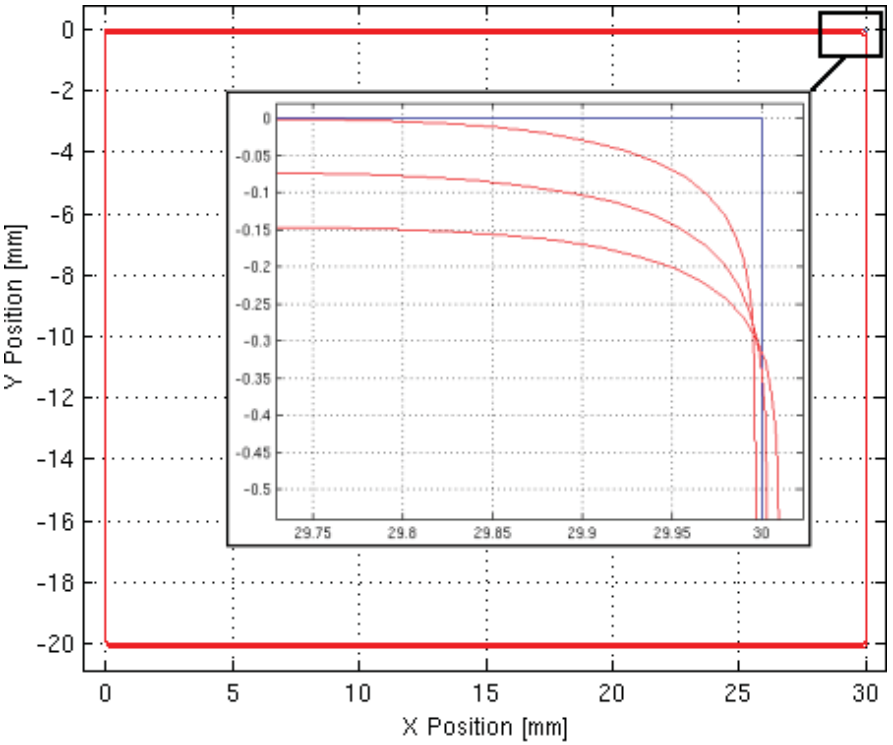


Figure 5.1 Diagram of x and y positions of the *RapidBox* reference (blue) compared to the processed reference (red).

Creating Path from Reference

Because of the reference deviation caused by the robot controller and the delay of samples when the robot was tracking trajectories, described in Section 4.2, a path was created in Matlab from the reference with intermediate points. This path

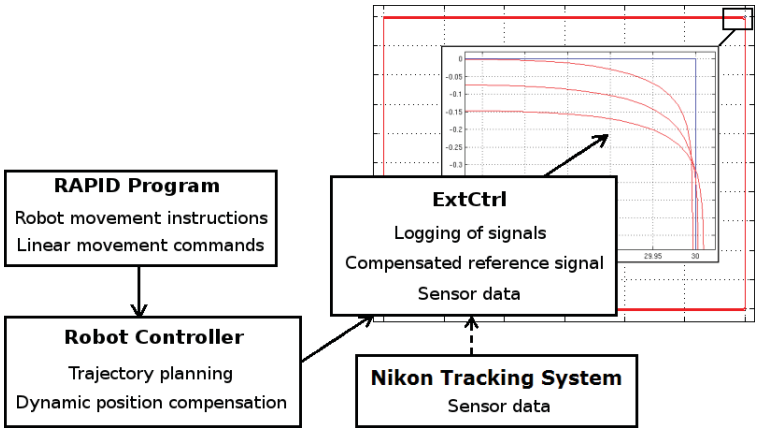


Figure 5.2 Illustration of how the data signals were processed before they were logged and monitored through ExtCtrl.

was then tracked by the robot to evaluate its position accuracy. By changing the number of intermediate points in the path, it could be aligned with the measured positions without deviations. The path was based on the *RapidBox* tool path that was created directly in RAPID code. It was used as a reference when creating ILC compensated paths, which were then translated into RAPID programs. Given below, are the commsnds used in Matlab for generating the reference path.

```
% CREATING PATH REFERENCE:

% Setting Dimensions for Rectangle:

xLength= 30;
yLength= -20;

% Value for Creating Intermediate Points:
step= h

% Create All Sides Of Rectangle:

x1P = 0: step: xLength;
y1P = zeros(1,length(x1P));

y2P = 0: -step: yLength;
x2P = xLength*ones(1, length(y2P));

x3P = xLength: -step: 0;
```

```
y3P = yLength*ones(1, length(x3P));

y4P = yLength: step: 0;
x4P = zeros(1, length(y4P));

% Creating Rectangle Sequence in Vectors:

xBoxPos = [x1P x2P x3P x4P];
yBoxPos = [y1P y2P y3P y4P];

% Creating Step Down Sequence in Z Direction:

z1P= 0: -step: -10;
z2P= -10: -step: -20;

% Creating Vectors to Time Robot Movement: Pause, In, and out:

zPause1= zeros(1, length(yBoxPos));
zPause2= -10*ones(1, length(yBoxPos));
zPause3= -20*ones(1, length(yBoxPos));

xyInOut = zeros(1,10);
zIn = xyInOut;
zOut = -20*ones(1,10);
xyPause = zeros(1,length(z1P));

% Creating Entire Sequence of Path in Vectors for X,Y, and, Z:

xPos = ([xyInOut xBoxPos xyPause xBoxPos xyPause xBoxPos
xyInOut])';
yPos = ([xyInOut yBoxPos xyPause yBoxPos xyPause yBoxPos
xyInOut])';
zPos = ([zIn zPause1 z1P zPause2 z2P zPause3 zOut])';
```

By setting the start values of the tool path to zero, the tool path could easily be added to any start position of the robot. To create a tool path with coordinates for the robot to follow, the reference was set up as vectors of x, y, and z put together in a sequence. The vectors were generated by setting a start and end value together with the value for the distribution of points. By setting the value of the parameter `step` (given as `h`) the number of points between the start and end value in the vector was set, also referred to as the resolution. Figure 5.3 shows a diagram of `xPos` (blue), `yPos` (red) and `zPos` (green), which was the sequence of vectors put together to create the tool path. The *Point Index* shows the values of x, y and z at each point.

Figure 5.4 shows the sequence of the x and y vectors that created the rectangle

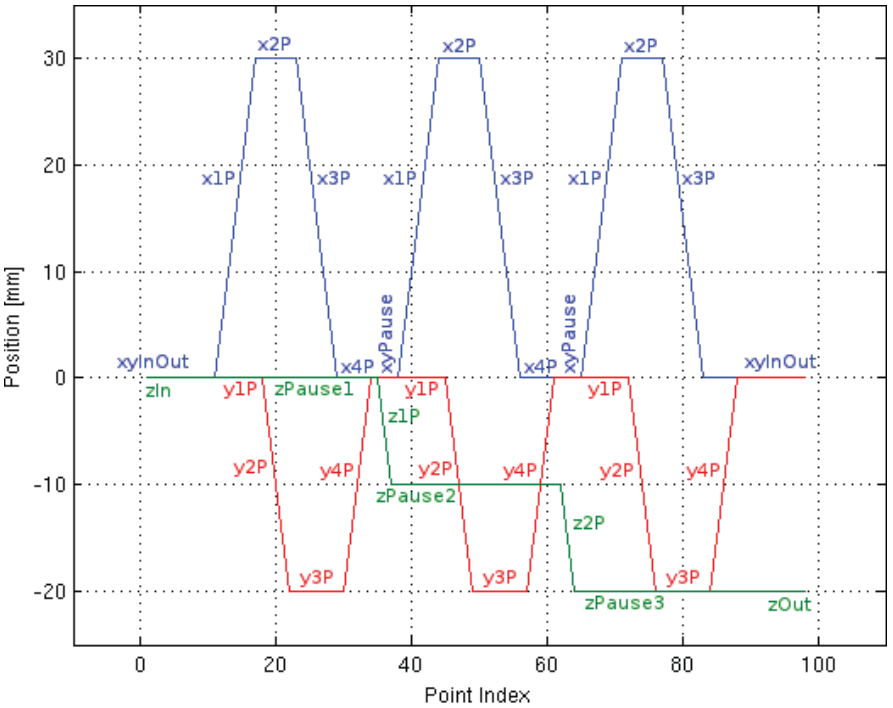


Figure 5.3 Diagram of `xPos` (blue), `yPos` (red) and `zPos` (green) showing the sequence of vectors generated in Matlab to create a tool path similar to *RapidBox*.

similar to the *RapidBox* tool path (Section 4.1). The points `y1P`, `x2P`, `y3P`, and `x4P` represent the vectors that had constant values.

Reference to RAPID Code

The original RAPID code program (*RapidBox*) was used as base when the new reference was translated into RAPID code. Given below is a simplified version of the *RapidBox* code that shows the coordinates for the initial move (`MoveJ`) and how the linear moves (`MoveL`) were changed to create the tool path that generates the rectangle. To see the whole *RapidBox* program, see Appendix B.1. The first three values of `MoveL` are the Cartesian coordinates at each point. These values changed in a sequence with the same magnitude as the reference generated in Matlab.

```
% Simplified RapidBox Program:

    MoveJ [[100.00,0.00,25.00], ...
```

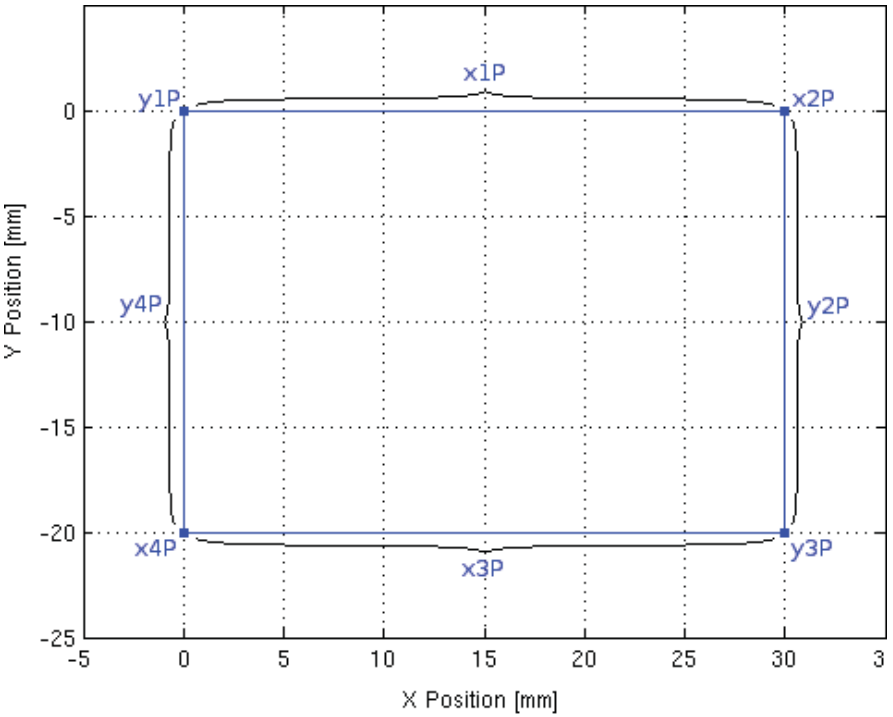


Figure 5.4 Sequence of the x and y vectors generated in Matlab that creates the rectangle similar to the *RapidBox* tool path.

```
MoveL [[100.00,0.00,25.00], ...
MoveL [[130.00,0.00,25.00], ...
MoveL [[130.00,-20.00,25.00], ...
MoveL [[100.00,-20.00,25.00], ...
MoveL [[100.00,0.00,25.00], ...
MoveL [[100.00,0.00,15.00], ...
...
MoveL [[100.00,0.00,5.00], ...
```

All values of rotations and orientations were kept fixed as well as the speed `v10` (10 mm/s) and `z0`, which forces the robot's tool center point to move to the exact point at each corner of the rectangle. The only values changing were the x, y, and z positions of the `MoveL` command, which simplified experiments and evaluation of the ILC compensation to only focus on the Cartesian translational moves.

By replacing all the Cartesian coordinates of the linear move commands (`MoveL`), the original program was kept but the positions changed according to the new reference. Each new position was given by `xPos(i)` and since

the values begin in zero the start positions were just added to each point, given from the initial position (`MoveJ`). In this case the start position was `MoveJ [[100.00,0.00,25.00], ... ]`. The code given below shows how the new positions at each point were inserted.

```
% Code to Generate Positions:

xPos(i) + 100
yPos(i) + 0
zPos(i) + 25
```

The Matlab code used in the script to translate the reference into RAPID code is given below. It created a txt file (`box_file.txt`) with all the `MoveL` commands that were inserted into the RAPID program to replace all the old positions.

```
% Code to Generate RAPID Program:

box_file = fopen('box_file.txt','w');

str = '';
for i = 1:2:length(xPos)
    str = [str, '\t', 'MoveL [[',
    num2str(xPos(i) +100, '%0.5f'), ', ',
    num2str(yPos(i), '%0.5f'), ', ',
    num2str(zPos(i) +25, '%0.5f'),
    '],[0.70708,8.94019E-06,1.40498E-05,0.707134]
    ,[-1,-1,0,1],* [9E+09,9E+09,9E
    +09,9E+09,9E+09,9E+09]], v10, z0,
    tdelcam1\\WObj:=wdelcam1;', '\n'];
end

fprintf(box_file, str);

fclose(box_file);
```

5.2 Creating ILC Path

The RAPID program, created from the path reference, were executed on the robot and all measured Nikon positions were logged. This data were then read into the Matlab and compared to the original reference. In this section it is described how the position error was added to the reference and a new ILC compensated tool path, based on the equations in Section 2.2, was created. The whole script used for analyzing and creating ILC tool paths is given in Appendix B.2.

Filter Design

As described in Section 2.2, a filter is needed to get rid of noise and high-frequency oscillations that the robot should not compensate. A low-pass Butterworth filter was therefore designed to filter the Nikon data. The Matlab command `[b,a]=butter(N,Wn)` was used to design a `N`th order Butterworth low-pass filter with the cut-off frequency `Wn`. With the command `filtered_data = filtfilt(b,a,raw_data)`, a zero-phased digital filtering was done on the the raw data by the previously designed filter. By testing different values for the `Wn` parameter the intensity of the filter was decided. The filtered signal should smoothly follow the unfiltered signal without bias. Figure 5.5 shows the result of the signal filtered by a 6th order lowpass filter (`N=6`) with the cutoff frequency `Wn=0.064` Hz.

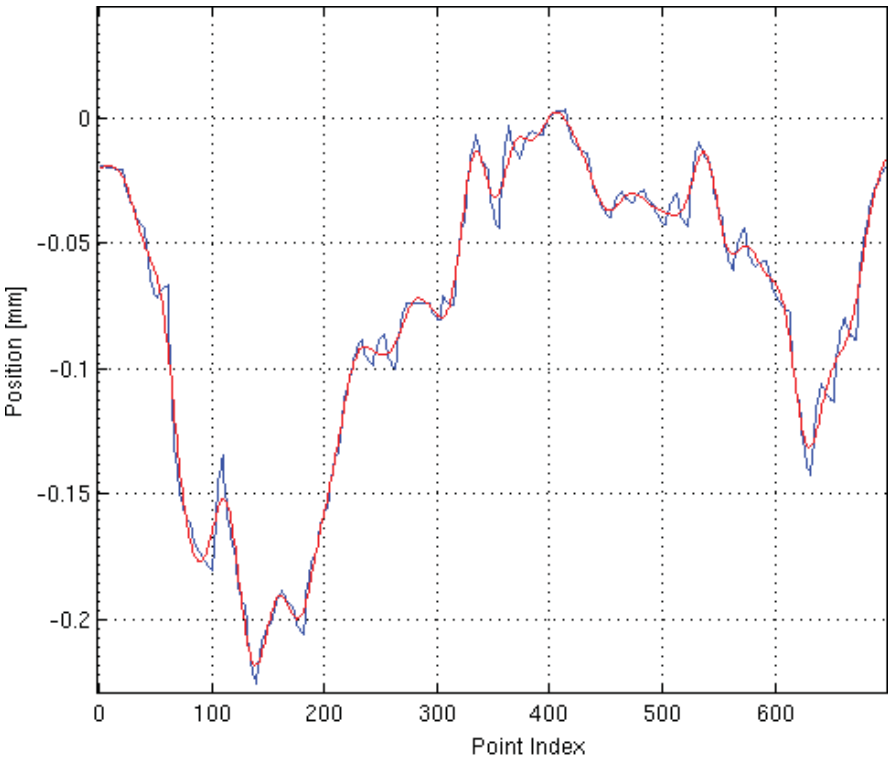


Figure 5.5 Diagram of filtered signal (red) of the Nikon data (blue) with a 6th order lowpass filter [250 Hz].

Aligning Reference

To create an ILC compensated tool path the position error between the reference path and Nikon data, calculated at each point in the program, must be added to the corresponding point. To do this compensation the number of data points must be the same in both the Nikon data and the reference. They also need to be aligned. If the data sets were not aligned the positions would be compared at different stages of the program giving an incorrect position error. This false error would then be added to the reference to create an incorrect ILC tool path. By changing the resolution of the reference with the parameter `step` and the start and end value of the Nikon data, the data sets could be aligned. With different values of `step` the resolution of the reference changed its appearance resulting in a position error with varying magnitude.

Some of the data sets also had an offset between the Nikon and robot coordinate systems. These offsets had to be compensated before the references could be aligned. To remove the offset, the average position error in x, y, and z was calculated and compensated in each data set. Figure 5.6 illustrates an example of a data set with an offset in x position. From parts of data where the reference was constant (green), the average offset was calculated from the corresponding position errors $\delta X_{1...6}$. The parts of data used to calculate the offset relative to the x-axis can be seen in Figure 5.7, where the Nikon data was plotted over samples (*Point Index*). The complete data set (red) was plotted with the parts used for calculating the offset (blue). A closer view on one of the parts of data used for calculating the offset can be seen in Figure 5.8. In Appendix B.2, the code for selecting parts of data can be seen.

Different values of `step` were plotted to choose the value with best matching curves. The `step` parameter was sensitive to small changes making the choice of value for `step` quite delicate. To illustrate these changes, diagrams of the reference with different values of `step` were compared to the Nikon data. Figure 5.9 shows a diagram where the reference (black) was not aligned with the Nikon data (red). With `step=1/24` the value got too big, which gave the reference a lower resolution. This step value made it faster than the Nikon data, resulting in an increasing position error (green) up to about 8 mm. The increasing error was caused by the fact that the plotted data started in the same position and then according to the resolution of the reference the displacement increased over samples (*Point Index*). If the curves were aligned the position error should be constant since they would not vary over time.

In Figure 5.10 the value for `step` was lower with `step=1/26`, which gave it a higher resolution. This resolution made the reference (black) too slow compared to the Nikon data (red), which resulted in an increased position error (green) up to about 14 mm. With `step=1/24.82` the reference was aligned with the Nikon data resulting in a constant position error up to about 0.4 mm, which is shown in Figure 5.11.

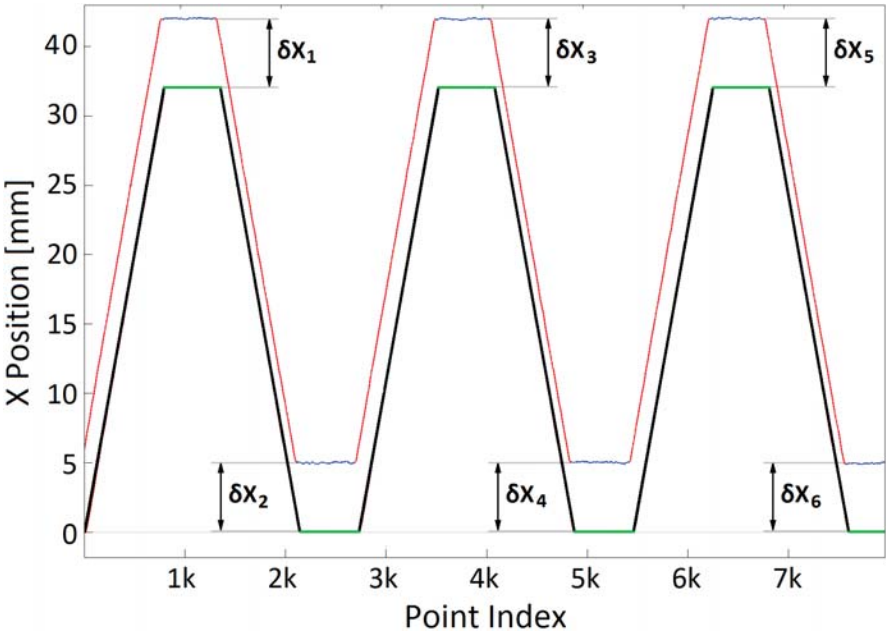


Figure 5.6 Example of a measured data set (red and blue) with an offset to the reference (black and green), plotted in x position over samples (*Point Index*) [250 Hz]. The position errors $\delta X_{1...6}$ were calculated between the parts of data where the reference was constant (green) and the corresponding measured data (blue).

Calculating ILC Tool Path

With the reference aligned to the filtered Nikon data, the ILC compensated tool path could be calculated. The position error was calculated for x, y, and z at each point of the program and was then added to the reference at that point.

To calculate ILC positions, the updating equation, given in (2.14) in Section 2.2 was used. Since the robot was following a path rather than a trajectory, the control signal was calculated for each point instead of time. The generalized ILC algorithms, presented in (2.15) were implemented in the ILC script.

By filtering the Nikon data with the low-pass Butterworth filter in Section 5.2, the discrete time filter $L(q)$, described in Section 2.2, was implemented. In (2.15c) and (2.14) the position error is filtered before it is added to the input. The measured signal (Nikon data) was, however, filtered when the ILC tool path was calculated. The position error was calculated from the original reference path, created in the ILC script in Section 5.1, and was therefore not filtered. Given below is the code for filtering the Nikon data in x positions (`xNikPosRaw`).

```
% Zero-phased Digital Filtering of X:
```

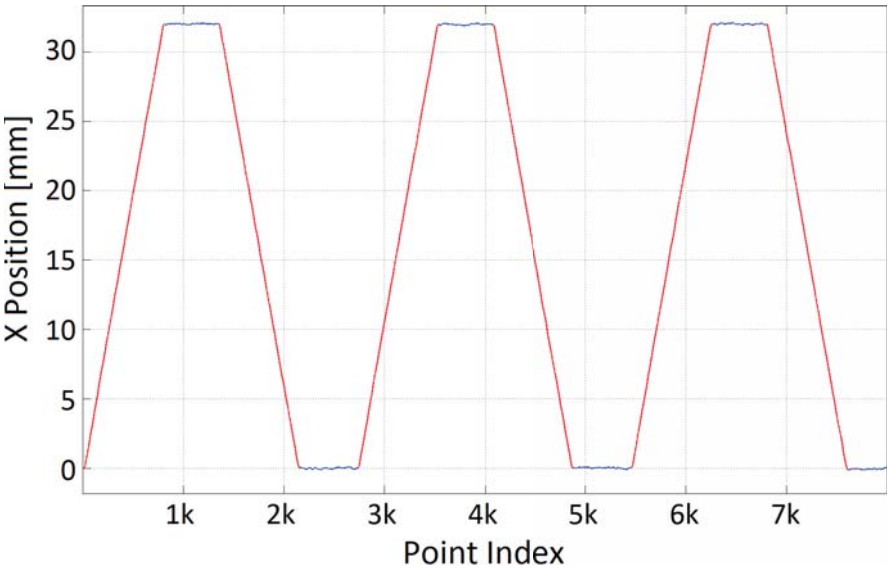


Figure 5.7 Overview of the complete data set (red) and data used for calculating offset (blue) plotted in x position over samples (*Point Index*) [250 Hz].

```
xNikPos = filtfilt(b, a, xNikPosRaw);
```

Based on (2.15a), the error was calculated between the reference (`xPos`) and the filtered Nikon data (`xNikPos`) with the code given below.

```
% Calculating Error in X Position:
xError(i) = xPos(i) - xNikPos(i);
```

This error (`xError`) was then added to the reference to create the updated control position (`xILC`) by the code given below, implemented from (2.15c).

```
% Calculating Updated Control Position for X:
xILC(i) = xPos(i) + xError(i);
```

The code below was used to calculate all the updated ILC positions for all axes in the program.

```
% Calculating All ILC Positions for X, Y and Z:
for i = 1: length(xPos)
    xError(i) = xPos(i) - xNikPos(i);
    xILC(i) = xPos(i) + xError(i);
    yError(i) = yPos(i) - yNikPos(i);
    yILC(i) = yPos(i) + yError(i);
```

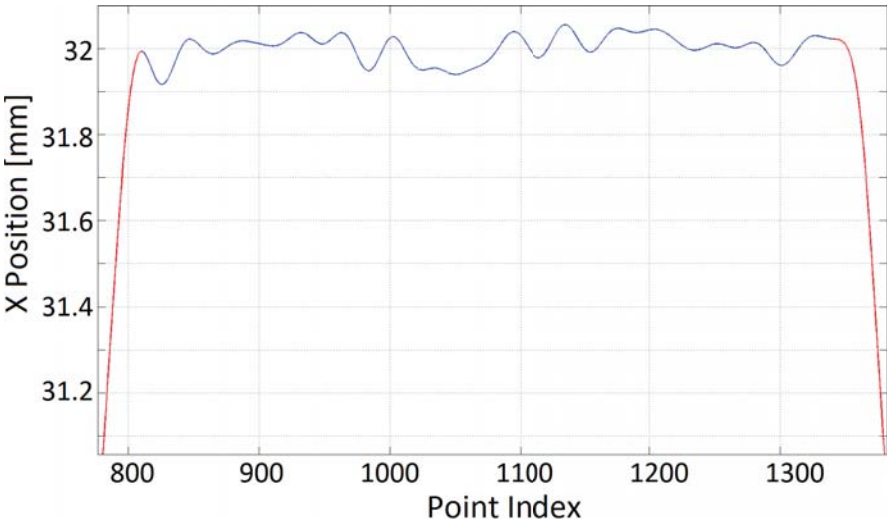


Figure 5.8 Diagram showing a close view of one of the parts of data used for calculating offset, seen in Figure 5.7, with data set (red) and data used for calculating offset (blue) plotted over samples (*Point Index*) [250 Hz].

```
    zError(i) = zPos(i) - zNikPos(i);
    zILC(i) = zPos(i) + zError(i);
end
```

By plotting the calculated ILC reference together with the Nikon data, reference and position error the ILC compensation could be evaluated. Figure 5.12 shows a diagram with the reference (black), Nikon data (red), position error (green) and the ILC compensation (blue) calculated for z-axis with `step=1/24.82`. It shows that the ILC compensation was correct since it follows the reference with the error added from the inverted offset of the Nikon data. To evaluate the ILC compensation in the x-axis, a zoom on one of the areas in Figure 5.13 was added because it moves over a larger distance than z. In Figure 5.14 the ILC compensation in y is shown, also with a zoom. The figures show that the ILC positions were calculated the right way for the three dimensions and that they could be used to create an ILC tool path. To generate the ILC program the compensation was added to the nominal path at each point along the path with the code given below and then automatically translated to `MoveL` commands.

```
% Code to Generate ILC Positions:

xILC(i) + 100
yILC(i) + 0
```

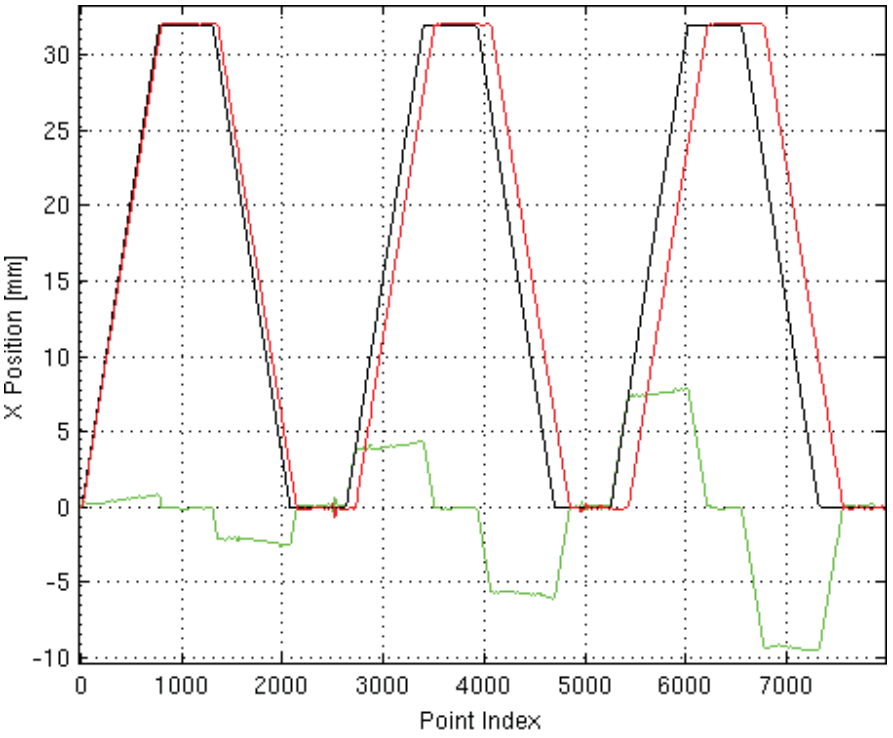


Figure 5.9 Reference (black) with `step=1/24`, Nikon data (red) and position error (green).

```
zILC(i) + 25
```

5.3 ILC Milling

To perform ILC milling experiments, pockets were milled out of the aluminum block with 1 mm left on each side for the generated program to mill. This pre-milling is referred to as roughing and the generated tool path performed the finishing of edges. With a tool diameter of 6 mm the roughing tool path was designed to remove all material inside the pocket, leaving 1 mm on the edges for finishing. To avoid the tool from hitting the bottom surface, the depth of the pocket was milled to 5 mm. The pockets are shown in Figure 5.15, where also the positions of the tool and spindle relative to the aluminum block are shown.

Figure 5.16 illustrates how the tool (orange) and tool path (red) was placed relative to the the pocket (dark grey) with edges marked in blue. At the center of the 6 mm diameter tool was the tool path which was created to perform a 1 mm

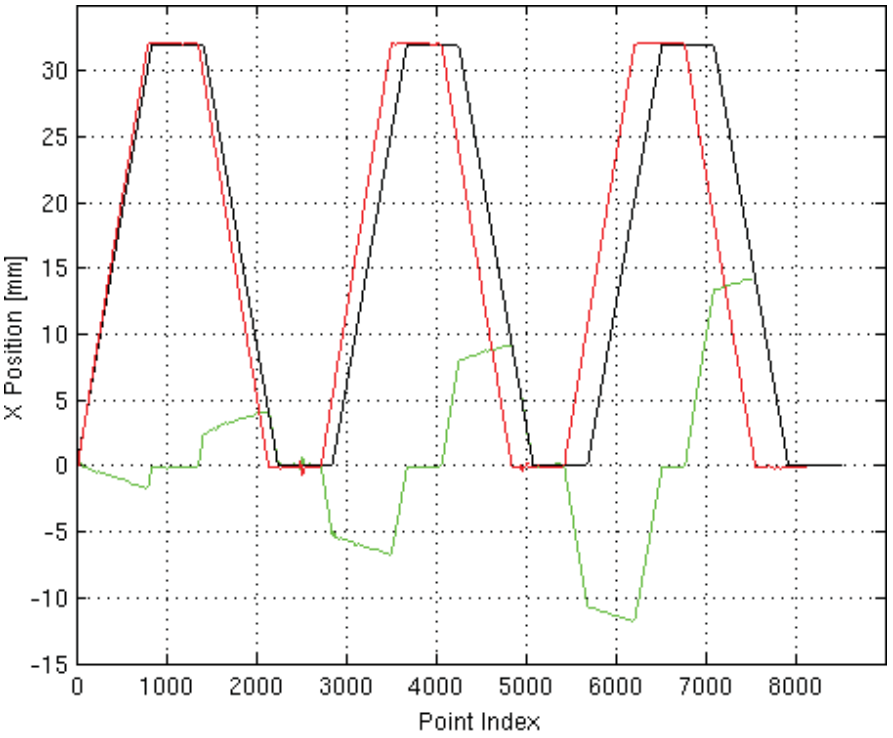


Figure 5.10 Reference (black) with `step=1/26`, Nikon data (red), and position error (green).

milling of the pocket edges. The finishing program was created to mill a total depth of 3 mm in z direction, with a 1 mm step down. Figure 5.17 shows a side view (z direction) of the pocket (dark grey) with the tool (orange) and the tool path (red) that makes the step down with a starting point in 0. The starting position of the tool path in z direction was set at 1 mm below the top surface of the pocket. This start position made the robot do an initial move that plunged down 1 mm to the starting point where it waited for the program to be executed.

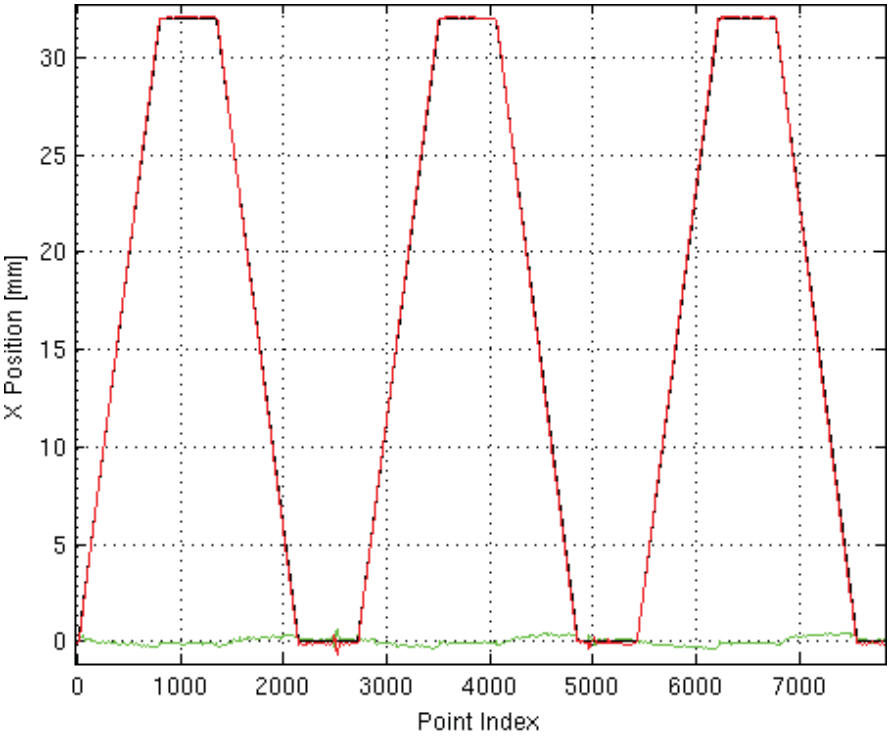


Figure 5.11 Aligned reference (black) with `step=1/24.82`, Nikon data (red), and a constant position error (green).

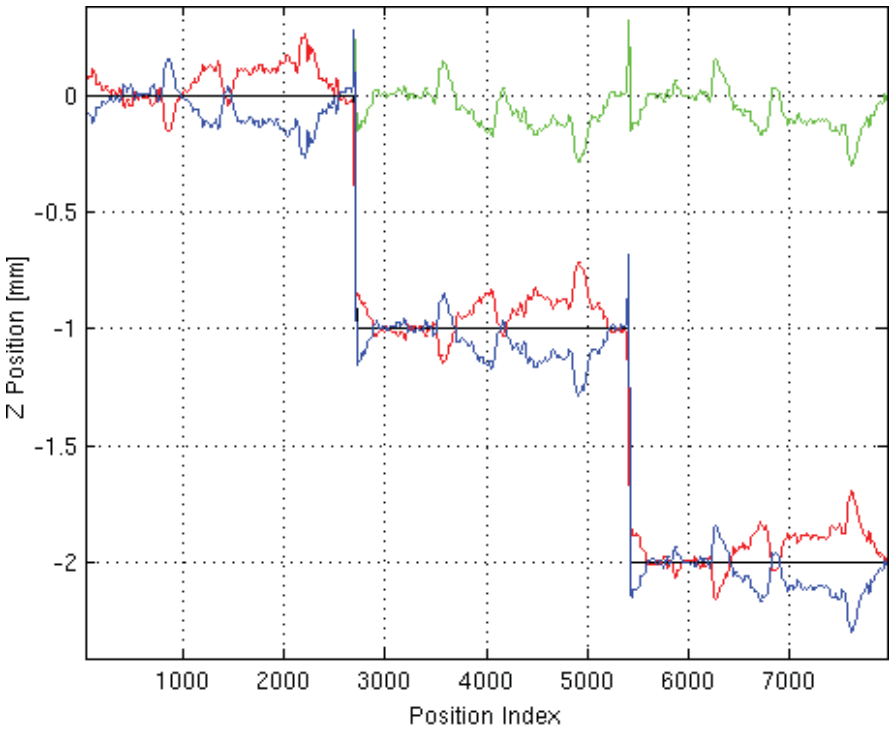


Figure 5.12 ILC compensation (blue), Nikon data (red), position error (green) and reference (black) with `step=1/24.82`.

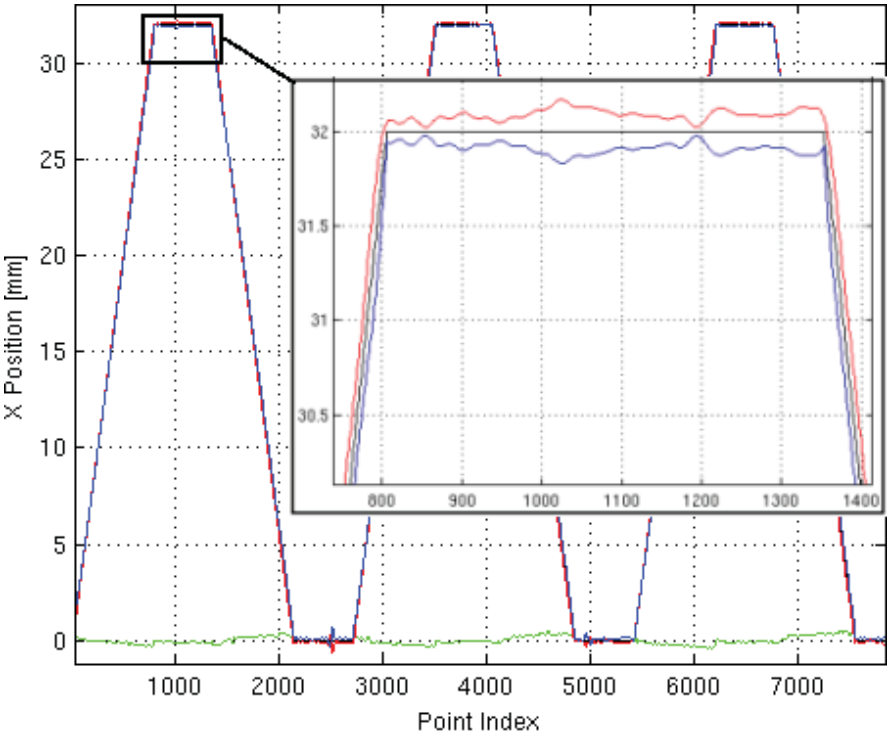


Figure 5.13 ILC compensation (blue), Nikon data (red), position error (green) and reference (black) with `step=1/24.82`.

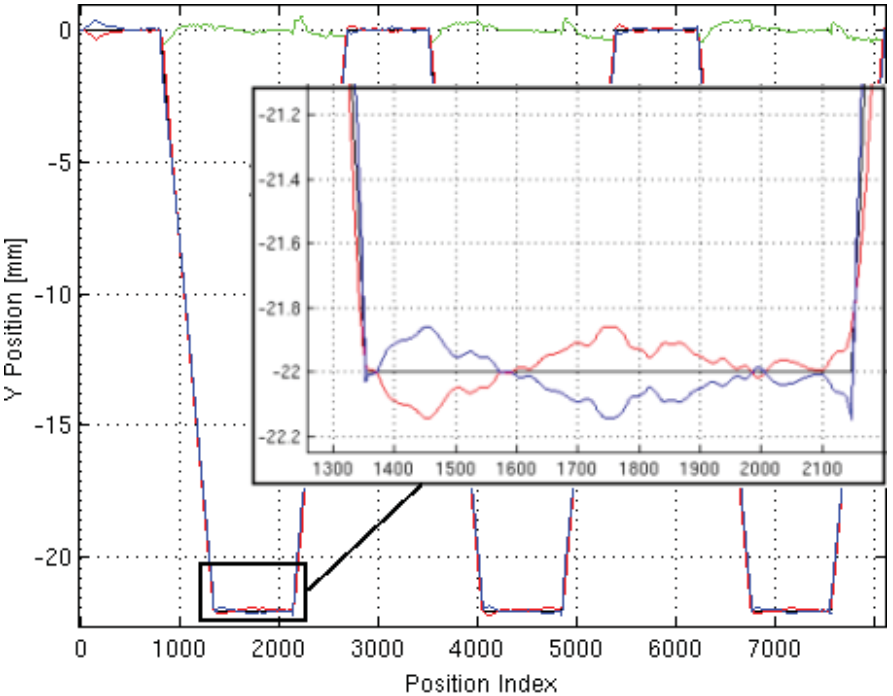


Figure 5.14 ILC compensation (blue), Nikon data (red), position error (green) and reference (black) with `step=1/24.82`.

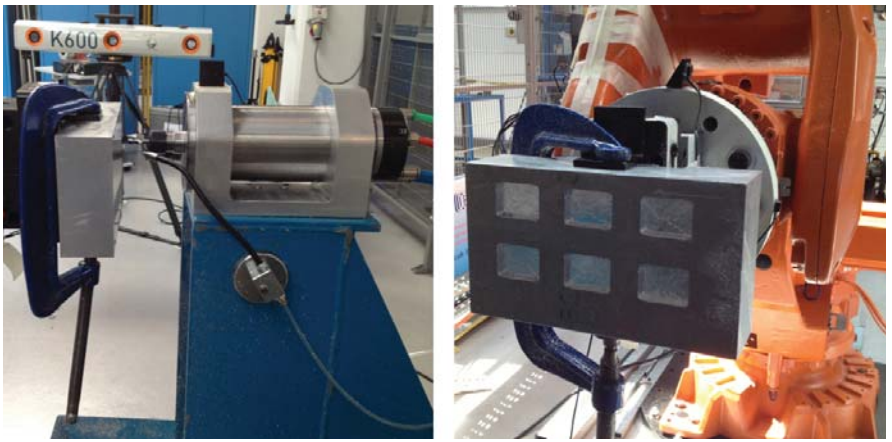


Figure 5.15 Placement of aluminum block and robot arm relative to tool and spindle.

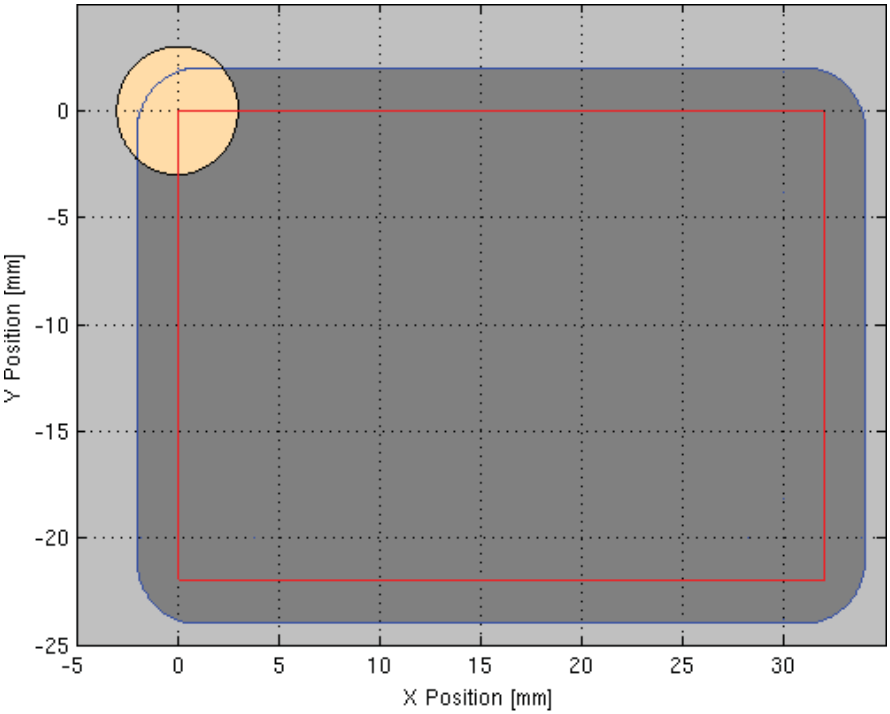


Figure 5.16 Top view of pocket (dark grey) with edges (blue), tool (orange), and tool path (red).

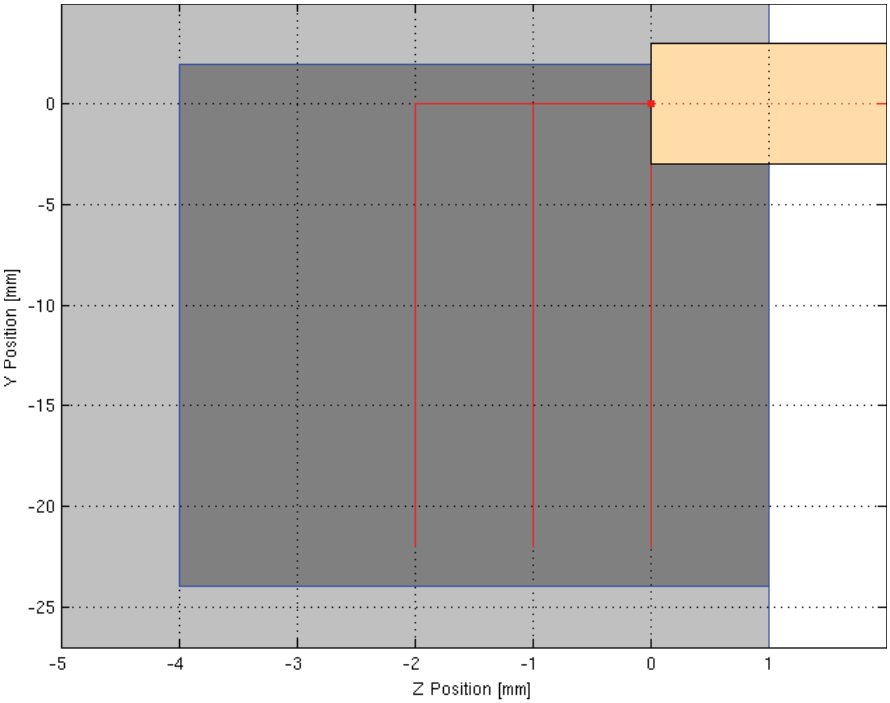


Figure 5.17 Side view of pocket (dark grey) with edges (blue), tool (orange), and tool path (red).

6

Results

An ILC compensated tool path was created for the robot to perform a task with a higher precision than before. To evaluate the result of using ILC, the measured positions that were logged from executions run with the nominal (uncompensated) path and the compensated ILC path were compared to the original reference. Experiments were done both with the robot moving in free space and when milling to evaluate the effects of disturbances caused by the milling.

6.1 Only First Iteration ILC

As described in Section 5.2 the reference must be aligned with the Nikon data to create the first iteration ILC tool path. For the same reason it requires that new Nikon data from subsequent tool paths can be aligned with the old for the following ILC iterations to work. The resolution of the measured data can not be changed in retrospect and therefore it requires that they can be aligned by only changing the starting point of data to be analyzed. If the measured Nikon data sets are unable to be aligned, the new position errors can not be added to the previous ILC tool paths at the correct point in the ILC program, which prevents further iterations of ILC tool paths.

After running first iteration ILC tool paths the new Nikon data were compared to the uncompensated data. The indexing of iterations, described in Section 2.2, was used for referring to the uncompensated data (*iteration:0*) and ILC compensated data (*iteration:1*). It showed that the Nikon data from first execution could not be aligned with the data logged from the ILC compensated path. The data from iterated paths did not match even if they were logged with the same robot speed, which can be seen in Figure 6.1 where both data sets were logged with a robot speed of 10 mm/s. In the beginning of the diagram the Nikon data were aligned due to a correct start point but then drift apart over time. By looking at the peaks in Figure 6.2, numbered 1-3 in Figure 6.1, the displacement of data can be seen. It shows that in the beginning of the diagram at peak 1 the data were aligned but at peak 2 the data were separated. At peak 3 the data have drifted even further apart.

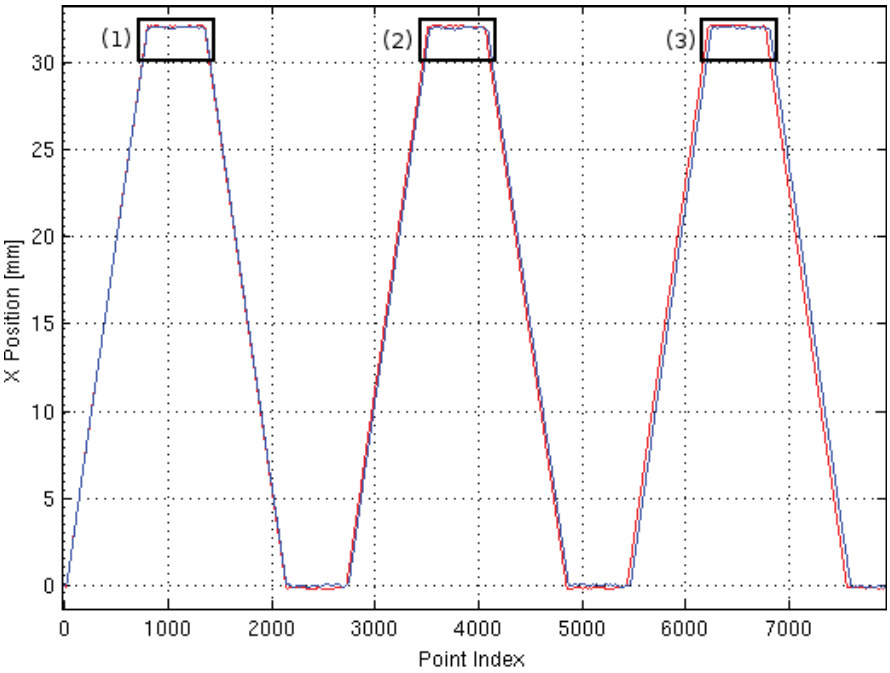


Figure 6.1 ILC *iteration:1* data (blue) compared to uncompensated *iteration:0* data (red) with peaks numbered 1-3.

The reason for the fact that the data could not be aligned was probably caused by the ABB IRC5 robot controller's interpretation of `MoveL` commands. When the robot was set to move linearly from one point to another the controller generates a trajectory between `MoveL` commands, which can have varying length or resolution. This trajectory generation effects the data length and causes the data between executions to vary in resolution.

Since the measured data could not be aligned, only first iteration ILC tool paths could be created. Executions from first iteration ILC tool paths were used in the following results to analyze the effects of using ILC when machining with robots. If the results after first iteration ILC were positive it was an indication that even further iterations might improve the results.

6.2 Free Space Results

By evaluating the results of ILC when the robot was moving in free space, the performance could be analyzed without disturbances that may occur when milling, such as vibrations. To analyze the exact result of ILC, the reference and measured

Table 6.1 Results of ILC in free space.

ILC Iteration:	**0:** [x, y, z]	**1:** [x, y, z]
Standard Deviation [mm]:	[0.1140, 0.0672, 0.0854]	[0.0495, 0.0476, 0.0281]
Reduced Error:		[56.55%, 29.13%, 67.11%]

data were plotted as x and z versus y. These diagrams eliminates the risk of analyzing false errors caused by misalignment of the reference.

The tool path used for evaluating the result of ILC in free space had the robot speed `v10` (10 mm/s). For the ILC tool path, `step=(1/24.82)` was used for aligning the reference. Figure 6.3 shows an overview of the rectangle diagram where x was plotted versus y. It shows the reference (blue), uncompensated *iteration:0* data (red), and ILC *iteration:1* data (green). A zoom on the left side illustrates the result in a closer look.

To quantify the position error from using ILC in free space, the variance and standard deviation were calculated for x, y, and z. These values were calculated from the same parts of data that were chosen to calculate the offset, described in Section 5.2 and exhibited in Figure 5.7. When the reference values were constant along x direction, the reference values were changing along y (or z) direction, and vice versa. The robot's ability to keep a fixed position in one direction, when moving along another direction, was evaluated by calculating the position errors from parts of data with constant reference. In Appendix B.2, the code for calculating these results can be seen. The values from the uncompensated data (*iteration:0*) and ILC data (*iteration:1*) are given in Table 6.1. The values showed an improvement in standard deviation with ILC of 56.6% along the x-axis, 29.1% along the y-axis, and 67.1% along the z-axis.

In Figures 6.4 - 6.7 the results in x and y position, plotted along the y and x direction respectively, are shown with the acceptable error of ± 0.1 mm (grey) to illustrate the accuracy specification. The result of ILC in z position, plotted along the y direction is seen in Figure 6.8. In Figure 6.9 the ILC result in z position (green) and uncompensated *iteration:0* data (red) along y around 0 reference (blue) is shown.

All figures show an improvement from the ILC compensated data in x, y, and z position, where almost all positions were within the ± 0.1 mm critical area, which was the acceptable error when milling. The best results can be seen in z position, where all measured positions were within the specified error, which corresponds to the improvement value given in Table 6.1.

Since the step down in z direction was so small (1 mm) it was fast enough to be aligned with the uncompensated *iteration:0* data when plotted over samples (*Point Index*). By looking at Figure 6.10 where the result of ILC in z position was plotted over samples and Figure 6.11, which is a zoom of the result around 0 reference, it is clear that there was an improvement over the whole execution with ILC along the

Table 6.2 Results from milling with ILC.

ILC Iteration:	**0:** [x, y, z]	**1:** [x, y, z]
Standard Deviation [mm]:	[0.1063, 0.0797, 0.0810]	[0.0942, 0.0641, 0.0353]
Reduced Error:		[11.39%, 19.60%, 56.42%]

z-direction.

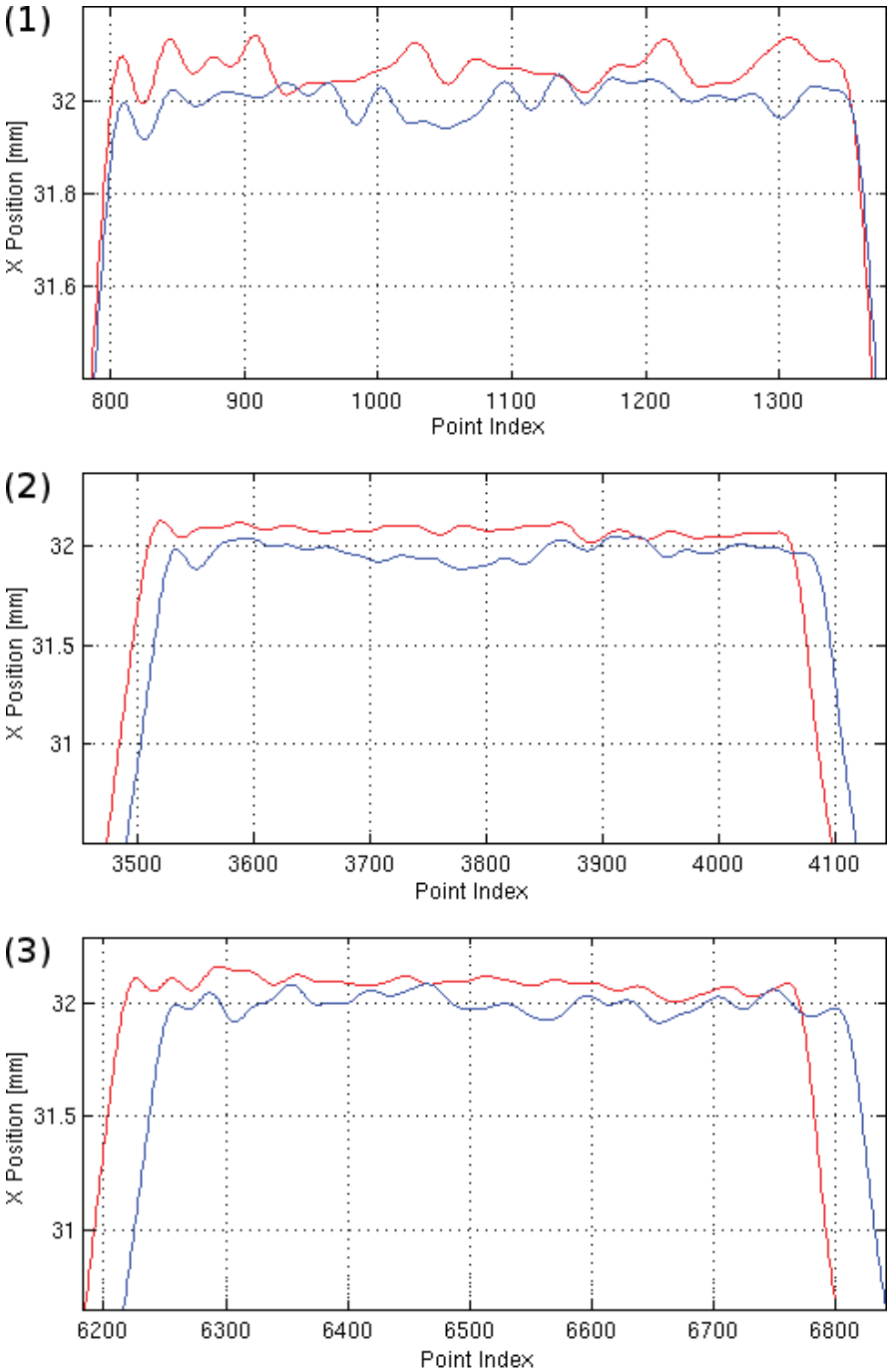
The results indicated an improvement in x, y, and z already after one iteration ILC compensation, which reduced the position error to about ± 0.1 mm. Since ILC indicated a significant improvement with ILC from moving the robot in free space, as seen in Table 6.1, it was encouraging to perform experiments with ILC for milling as well.

6.3 Milling Results

To evaluate ILC results when milling, one of the pockets was milled with the finishing reference described in Section 5.3 without compensation. It was milled with a robot speed of 10 mm/s and `step=(1/24.82)`. The data logged from this milling was then used for calculating the ILC compensation. By comparing the uncompensated data (*iteration:0*) to the data logged from the ILC compensated reference (*iteration:1*), the results were evaluated.

In Table 6.2, the calculated variance, standard deviation, and improved standard deviation are given. The improvement from milling with ILC was 11.4% in x-axis, 19.6% in y-axis, and 56.4% in z-axis. The results from milling with ILC in x and y position plotted along the y and x direction are shown in Figures 6.12 - 6.15. Figure 6.16 shows an overview of the result of ILC in z position along y. In Figure 6.17 the ILC result in z position (green) and uncompensated *iteration:0* data (red) along y around 0 reference (blue) is shown.

In Figure 6.18, the result of ILC in z position is plotted over samples (*Point Index*). A closer view of the result around 0 reference is seen in Figure 6.19, where it is clear that there was an improvement over the whole execution when milling with ILC along the z-direction.



80 **Figure 6.2** Peaks numbered 1-3 from data sets shown in Figure 6.1 with ILC *iteration:1* data (blue) and *iteration:0* data (red).

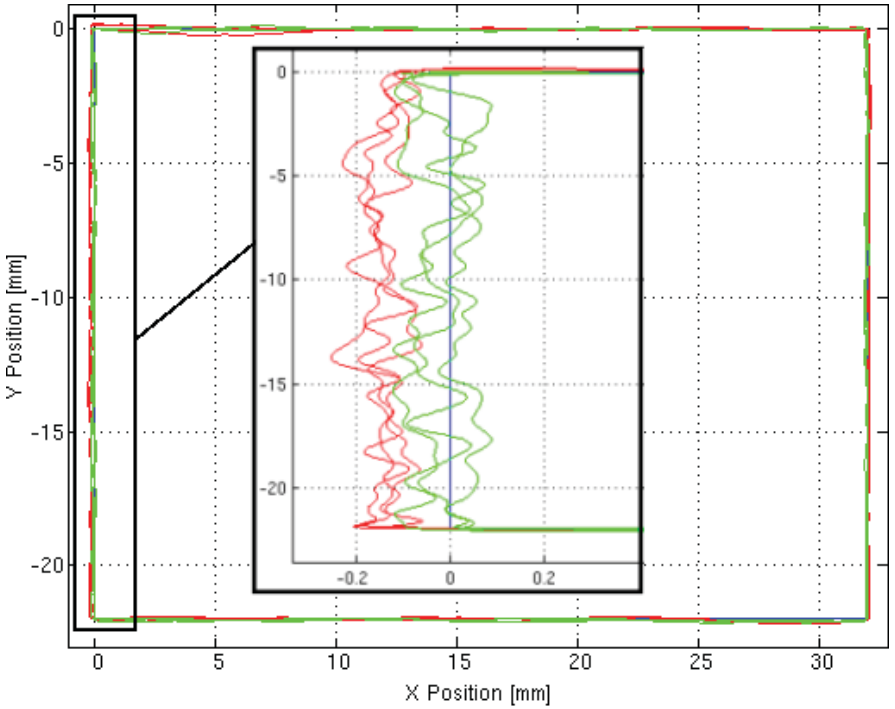


Figure 6.3 Overview of free space result with reference (blue), uncompensated *iteration:0* data (red), and ILC *iteration:1* data (green).

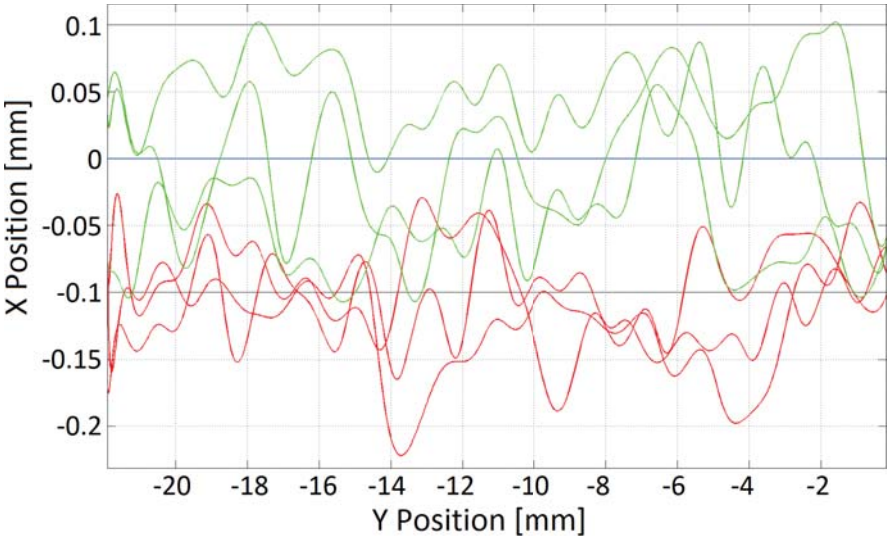


Figure 6.4 Free space result in x position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey).

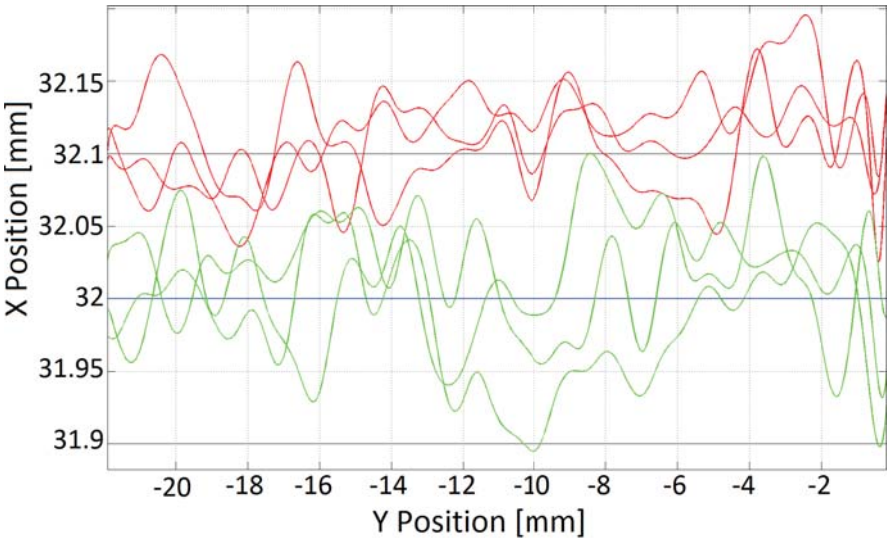


Figure 6.5 Free space result in x position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 32 reference (blue) with error limits (grey).

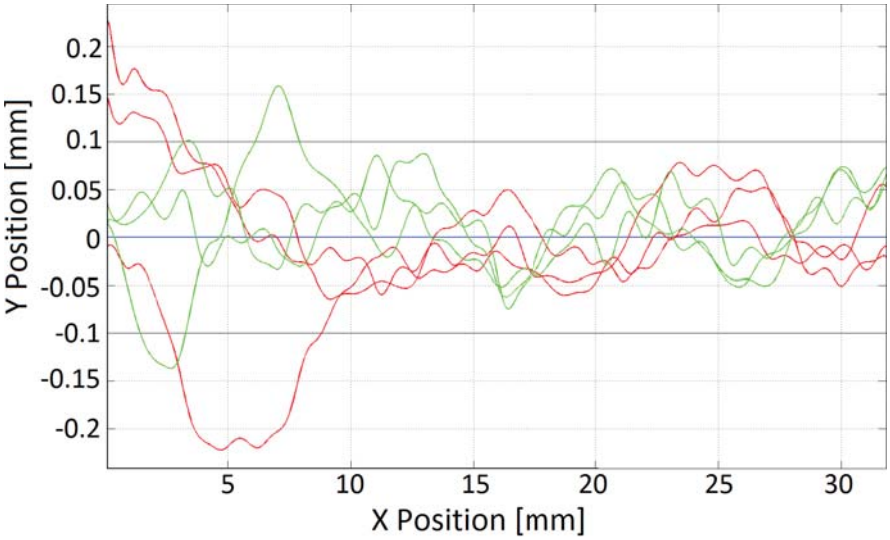


Figure 6.6 Free space result in y position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey).

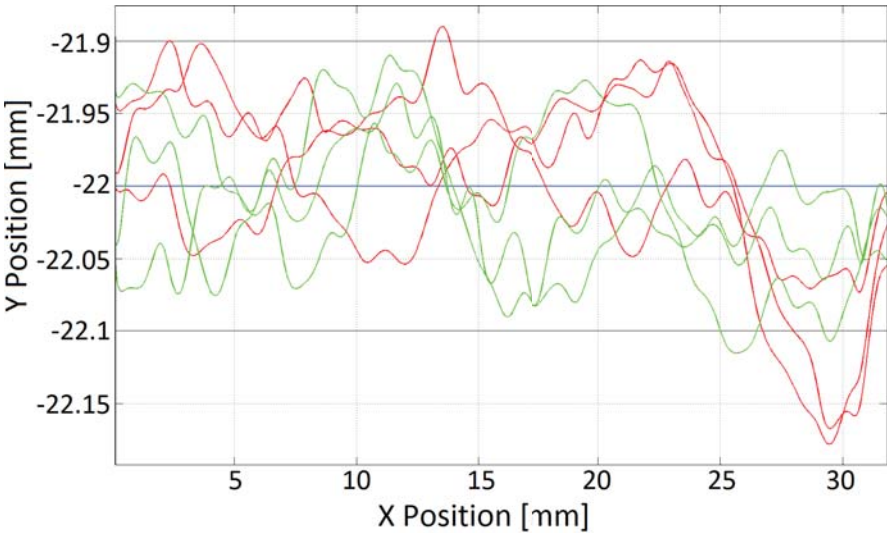


Figure 6.7 Free space result in y position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around -22 reference (blue) with error limits (grey).

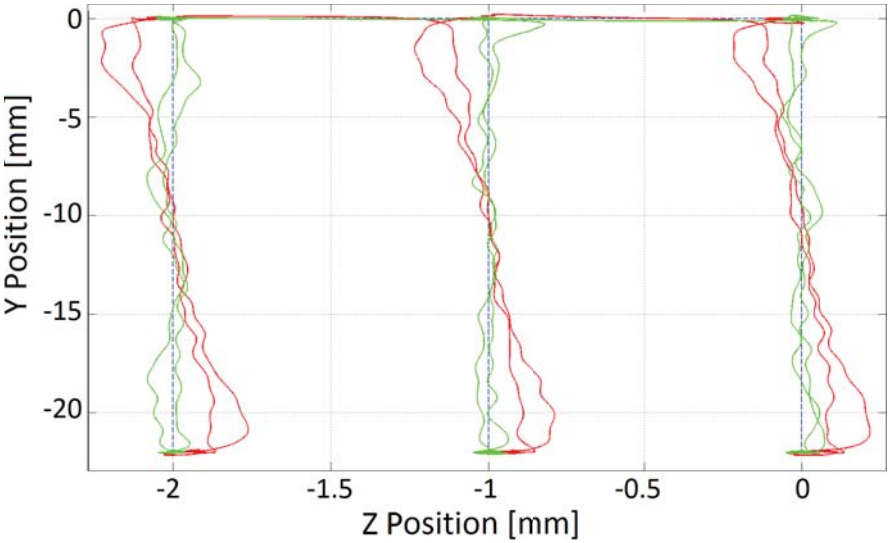


Figure 6.8 Overview of free space result in z position of ILC *iteration:1* (green), uncompensated *iteration:0* data (red), and reference (blue) with error limits (grey).

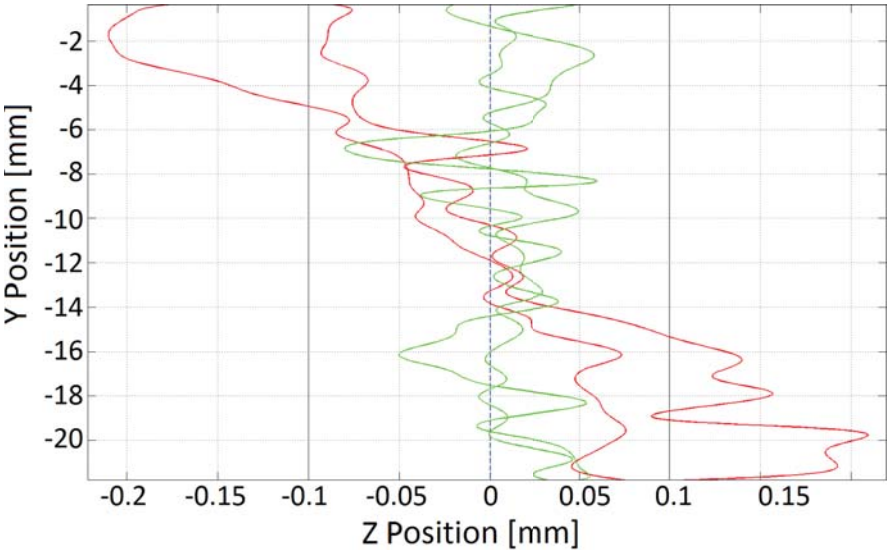


Figure 6.9 Free space result in z position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey).

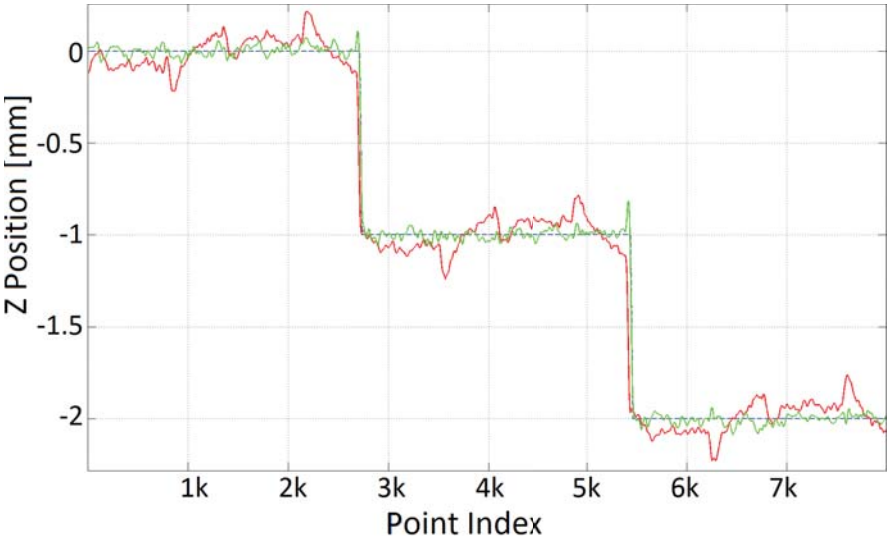


Figure 6.10 Overview of free space result in z position of ILC *iteration:1* (green), uncompensated *iteration:0* data (red), and reference (blue) over time (*Point Index*) [250 Hz].

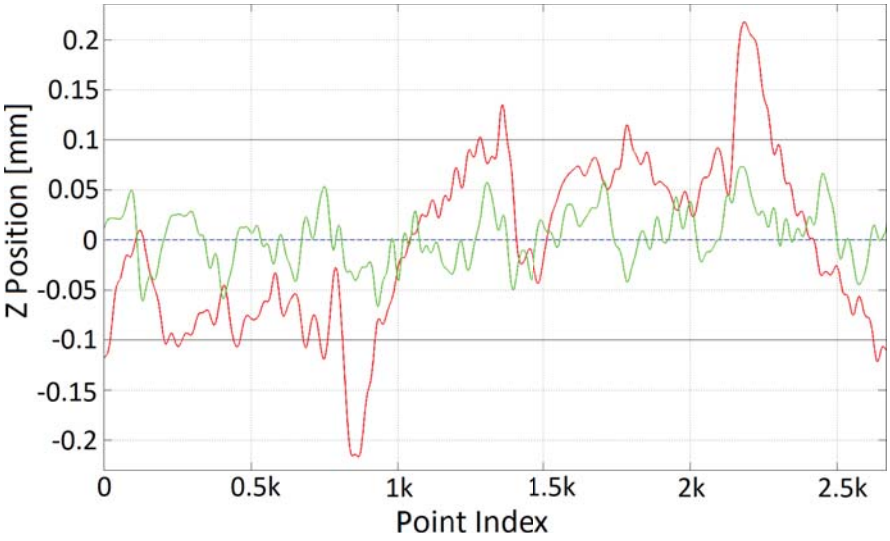


Figure 6.11 Free space result in z position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey) over time (*Point Index*) [250 Hz].

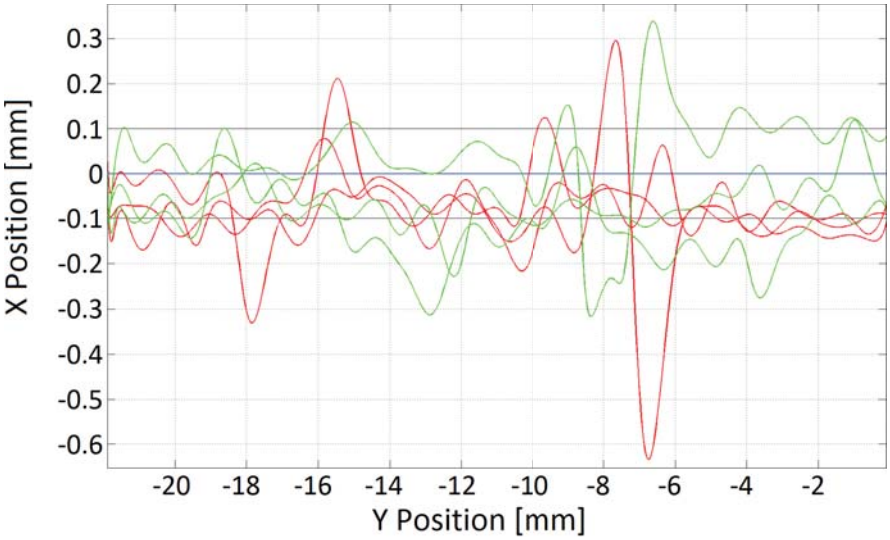


Figure 6.12 Milling result in x position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey).

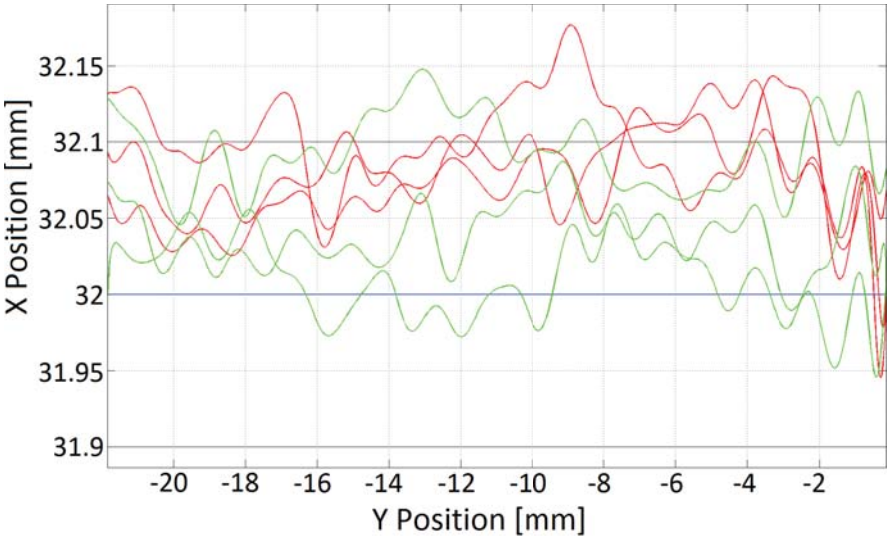


Figure 6.13 Milling result in x position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 32 reference (blue) with error limits (grey).

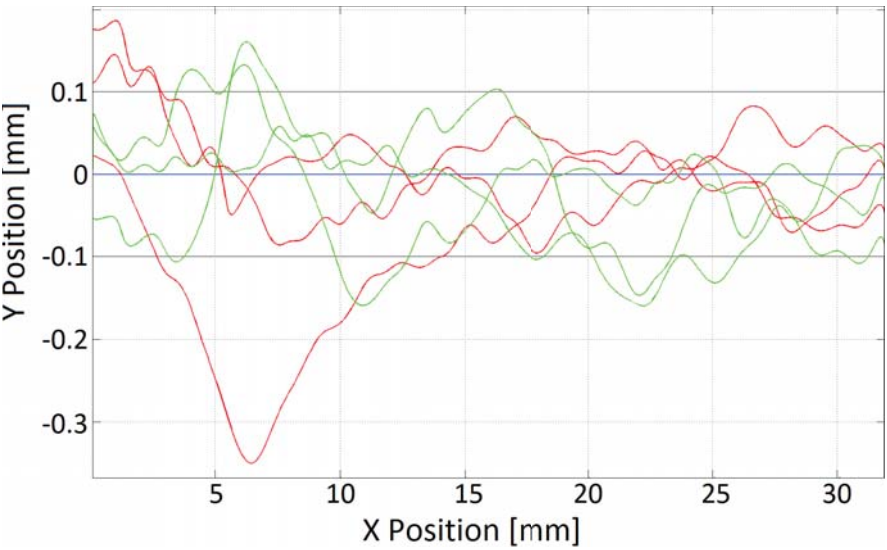


Figure 6.14 Milling result in y position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey).

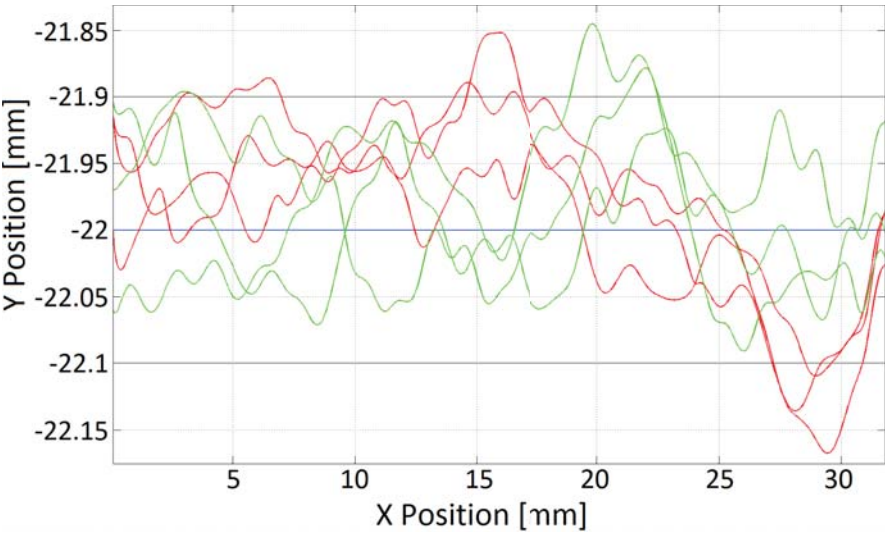


Figure 6.15 Milling result in y position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around -22 reference (blue) with error limits (grey).

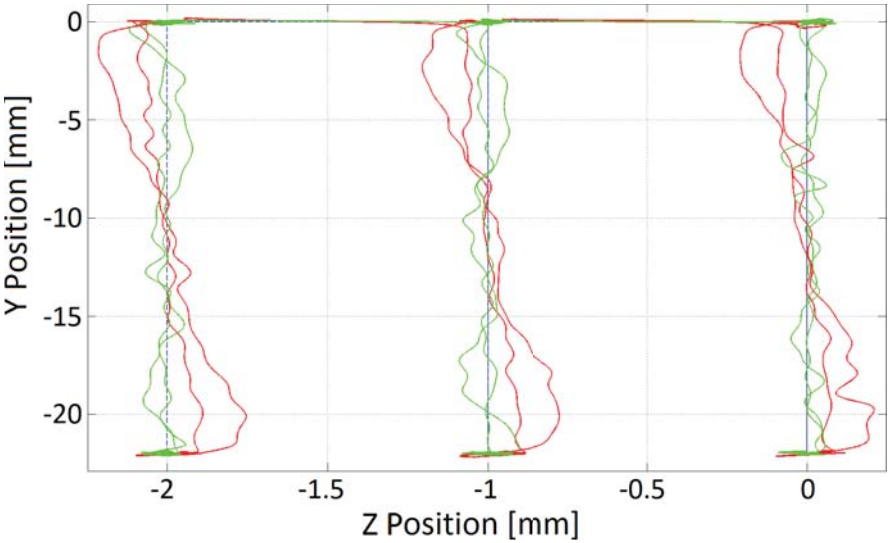


Figure 6.16 Overview of milling result in z position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around -22 reference (blue).

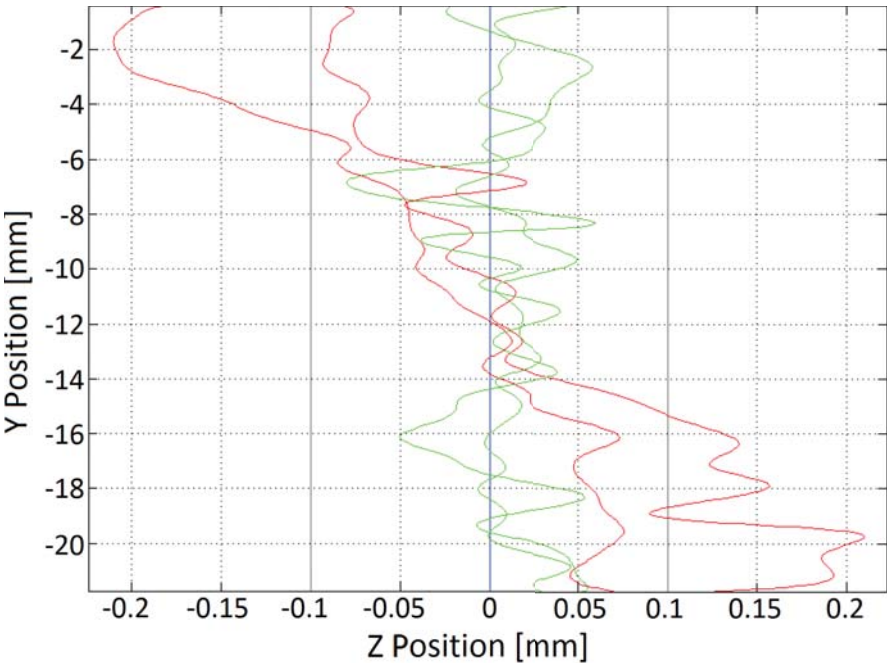


Figure 6.17 Milling result in z position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey).

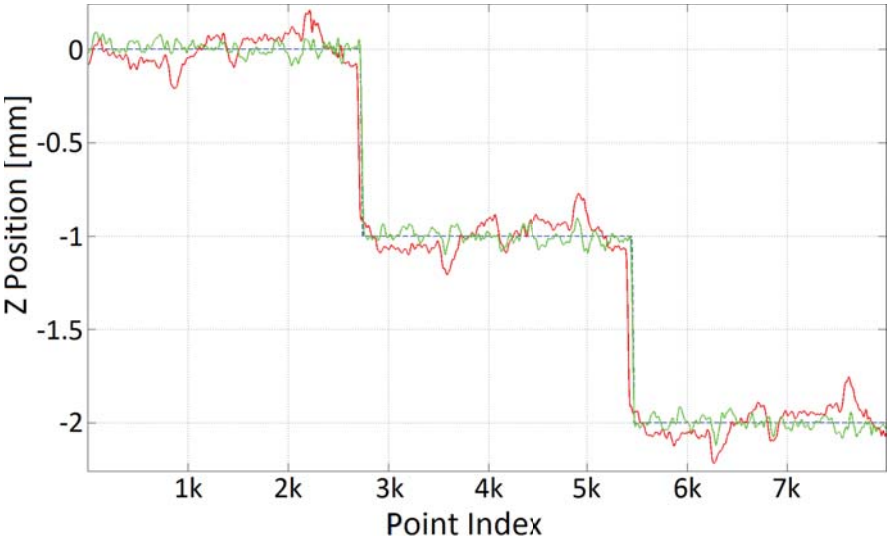


Figure 6.18 Overview of milling result in z position of ILC *iteration:1* (green), uncompensated *iteration:0* data (red), and reference (blue) over time (*Point Index*) [250 Hz].

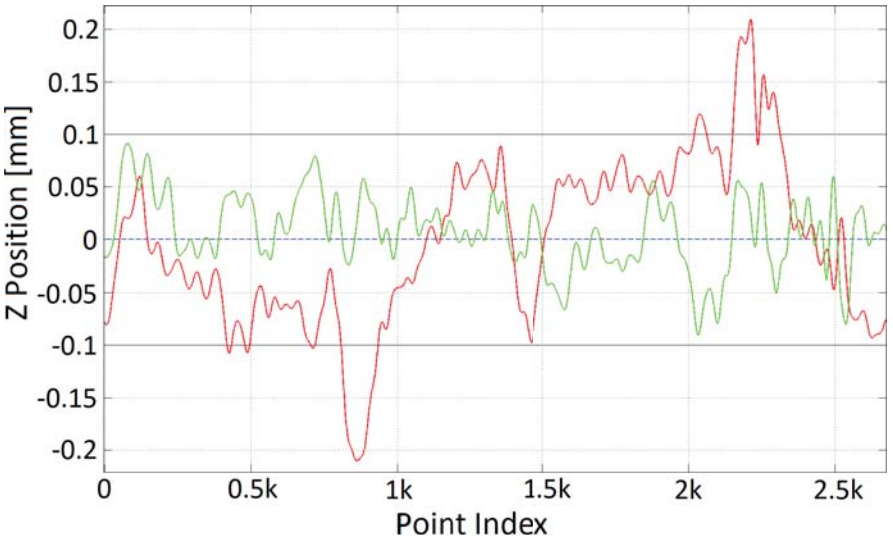


Figure 6.19 Milling result in z position of ILC *iteration:1* (green) and uncompensated *iteration:0* data (red) around 0 reference (blue) with error limits (grey) over time (*Point Index*) [250 Hz].

The result showed an improvement from ILC compensation also when milling, as given by Table 6.2. Diagrams with ILC milling results in z direction showed a significant improvement, which corresponds to the calculated improved standard deviation of 56.4%. By looking at the diagrams of the results from milling in x and y directions, it can be harder to see the improved results from milling compared to the free space diagrams. It is harder because the improved standard deviation was calculated as 11.4% for x and 19.6% in y, which can be harder to see by looking at the diagrams. The improved standard deviation was therefore used to show that there was an improvement in all directions when milling with an ILC compensated path.

6.4 Results from SIR

The ILC methods and scripts described in this thesis were used by SIR in Italy to perform experiments with the ILC approach. SIR had a similar robot cell as AML with the same robot and Nikon tracking system. When testing the robot's performance when tracking trajectories, as described in Section 4.2, they had a similar delay of samples [Berselli et al., 2013b].

SIR performed experiments with ILC when moving the robot in free space, the same way as described in Section 6.2. The ILC tool paths were calculated and analyzed by using the same Matlab script as described in Section 5.1. Measurements

from SIR can be seen in Figure 6.20, which shows the result of using ILC for free space motion.

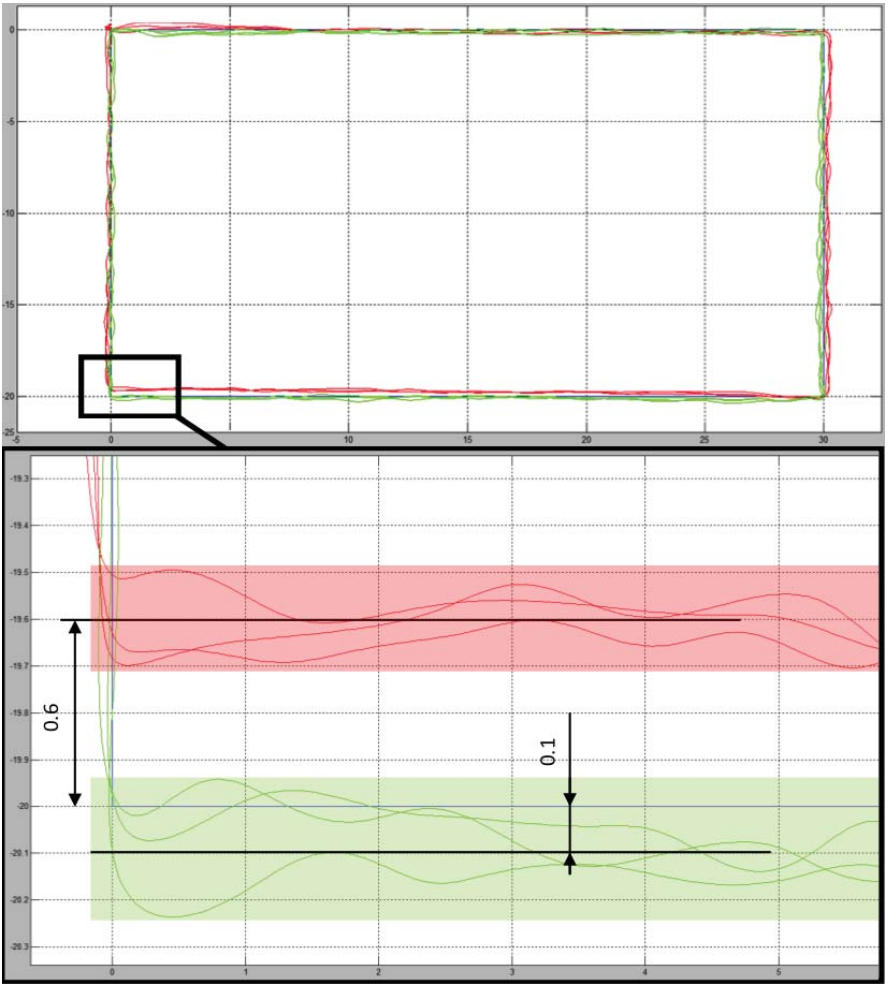


Figure 6.20 Results from SIR using ILC in free space, with ILC *iteration:1* (green), uncompensated *iteration:0* data (red), and reference (blue) [Berselli et al., 2013b].

The results showed an improvement from using ILC also at SIR. As seen in in Figure 6.20, the deviation from the reference was lowered from 0.6 mm to 0.1 mm. These results encouraged SIR to also perform experiments for milling with ILC [Berselli et al., 2013b].

7

Discussion

This chapter discusses the results in relation to problem definition and possible improvements. It also includes an analysis of the results achieved in this thesis.

7.1 Results in Relation to Problem Definition

To analyze the results presented in Chapter 6 they need to be put in relation to the requirements in Section 1.2. First of all, high precision manufacturing was performed with the robot both at AML and SIR. It was performed with an off-line feedback control system with an integrated high-accuracy sensor. Secondly, the implemented control strategy was based on ILC algorithms, which increased the robot's performance as described in the results. The script described in Section 5.2 was used to calculate ILC compensated tool paths in three dimensions. Experiments showed that there were improved results in all three dimensions when milling with ILC tool paths.

The results presented were achieved after milling with only one iteration ILC milling. The results show a reduced position error well within the specified tolerance of a maximum position error of ± 0.1 mm in z direction. The milling results were also improved for the x and y direction with between 11-20%. However, the position errors over the whole execution along the x and y direction were not within the specified tolerance. The idea of the ILC control strategy is that further iterations should be performed to further increase the results. Since there was such an improvement already after one iteration, the ILC control strategy seems to be successful and indicates that further iterations might get results within the specified tolerances for all dimensions.

7.2 Possible Improvements and Error Analysis

This section discuss possible improvements of the method used in this thesis and the error analysis. The latter aims to analyze the results achieved in the previous chapters.

Discussion on ILC Assumptions

In Section 2.2, where the theory behind ILC is presented, it was assumed that the initial conditions of the system was reset for every new iteration and that the system was time invariant. Since the robot was required to follow a path with high accuracy, without a time reference, the system was time invariant, assuming no tool wear and the same working volume. The initial conditions of the robot should also be reset to the same values between every iteration. For all executed iterations the speed of the robot was the same as well as the value for moving in corners. These assumptions were true for the free-space movement, where the start positions were the same between iterations. However, when performing milling experiments the start position, from where the tool path was executed, was moved in x and y position to mill the different pockets in the aluminum block, seen in Figure 5.15. This movement might have effected the result from milling with ILC. To reset all initial conditions the start position should be the same as well as the placement of the block for every iteration.

As mentioned before, another assumption for using ILC is that the tracking error was the same for every iteration as long as the input signal was the same. When looking at diagrams from milling with ILC, the compensated *iteration:1* data at some points appear with worse results that the *iteration:0* data, as seen in Figure 6.15. In the diagram it can be seen that the *iteration:1* data has a larger deviation than the *iteration:0* data around *X Position:20*. This deviation could be a result of disturbances that occurred at this particular position at a certain time. Further milling experiments should be done to evaluate if stochastic non-predictive deviations appear or if they can be compensated by ILC. If they do appear, an approach could be to combine the ILC method with an accelerometer that measures the vibration together with a frequency controller that adjust the spindle speed. Experiments from using a spindle frequency controller have been successful when tested at TEKS Ltd in Sheffield, UK, one of the other COMET partners [Berselli et al., 2013a].

To evaluate improvement from multiple executions, both the *iteration:0* and *iteration:1* paths should be repeated and compared to give an idea of the average improvement from using ILC. Because of time constraints and limited access to the robot at AML, multiple milling experiments were not performed.

Rotation of Z Reference

The result from milling with ILC showed an improvement after one iteration of approximately 11-20% in x and y direction and 56% in z direction. The best results were given in the z direction. This result, however, seems to be the effect of ILC compensating for a rotation in the calibration of the robot. This rotation can be seen in Figure 3.4, where the ABB teach pendant was showing values of rotation in Euler angles. It is likely that the robot was slightly rotated in the calibration, which would effect the reference sent to the robot. This rotated calibration might explain why the reference seen in Figures 6.8 and 6.16 seems to be rotated, which

was then compensated. It is probably why the improvement of ILC in z direction was calculated to be more than 50%.

A rotation in z position should appear in diagrams with the largest deviation at the beginning and an inverted deviation at the end, which can be seen in Figure 6.17. By looking at the mid section (*Point Index:1000-1750*) of the diagram in Figure 6.19, where z positions were plotted over samples, it can be seen that there was still an improvement with ILC, even though it might not be as much as 50%. To calculate the exact improvement, the reference should be rotated to match the rotation in Euler angles. This reference rotation would probably lower the improvement value. The result, however, shows that the ILC compensation works, even if it is compensating rotations or other kinematic calibration errors of the work piece or robot.

8

Conclusion and Future Work

This chapter summarizes the work and results described in this master thesis. It is a conclusion of how the results relate to the problem definition and the goal of this research project.

8.1 Conclusion

The results showed an improvement in all directions when milling with ILC. In Table 6.2, the resulting standard deviation and improved standard deviation with ILC are given. It shows an improvement of 11.4% for x, 19.6% for y, and 56.4% for z. These numbers were calculated from first iteration ILC compensation and indicate that even further iterations might improve the results. These results demonstrates the potential of using ILC.

SIR also presented successful results, based on the methods described in Section 5.1, that showed that ILC gave an improvement for their robot as well. These results verified that ILC worked when they were moving the robot in free space. It is an encouraging indication that ILC might work for SIR when milling as well.

With the results from this master thesis it was proven that when using ILC for milling with industrial robots in advanced manufacturing, the accuracy could be improved simultaneously in 3D. It gave a strong indication that by performing further ILC iterations, the robots would get accurate enough to produce parts within specified tolerances. It was an important milestone for the COMET project, which was extended to enter faces of optimizing and industrializing the solution. It is a further step into enabling cost effective, high precision manufacturing in factories of the future.

8.2 Future Work

This section summarizes the work needed to pursue the research in this master thesis.

Further ILC Iterations

To fully use the potential of ILC, experiments with further iterations should be performed. As described in Section 6.1, experiments were only performed with *iteration:1* ILC tool paths. There was a limitation of iterations because the data logged from different executed iterations could not be aligned, probably since the robot controllers generated trajectories between `MoveL` commands. A solution to this problem could be to bypass the robot control with the *obtain* mode, described in Section 1.3. This mode would force the robot to move exactly according to the path created in Matlab. By using the same amounts of positions for the iterations, the data sets would have the same resolution and could then be aligned. However, the *obtain* mode disables parts of the safety control in the ABB IRC5 controller, which makes sure that the robot does not make sudden fast movements over large distances. A safety would therefore have to be added to the Simulink model.

If the problem with delayed samples when tracking trajectories could be eliminated, the generalized ILC equations in (2.15), described in Section 2.2, are easier to implement, which makes path tracking easier. The sequence of calculated errors in (2.15a), could be added directly to the updated control signal in (2.15c). Errors from further iterations could then be added directly to the previous control signal, eventually converging the output to successful results.

Multiple Executions

To further evaluate the improvement from using first iteration ILC when milling with industrial robots, multiple executions of both the *iteration:0* and *iteration:1* tool paths could be performed, as described in Section 7.2. By comparing the results from repeated executions of the same tool path, an average of the improvement from using ILC could be calculated. The average can be used to analyze if the position errors were repetitive or affected by disturbances.

A

Simulink Models

A.1 Kinematic Model

A.2 Trajectory Compensated Model

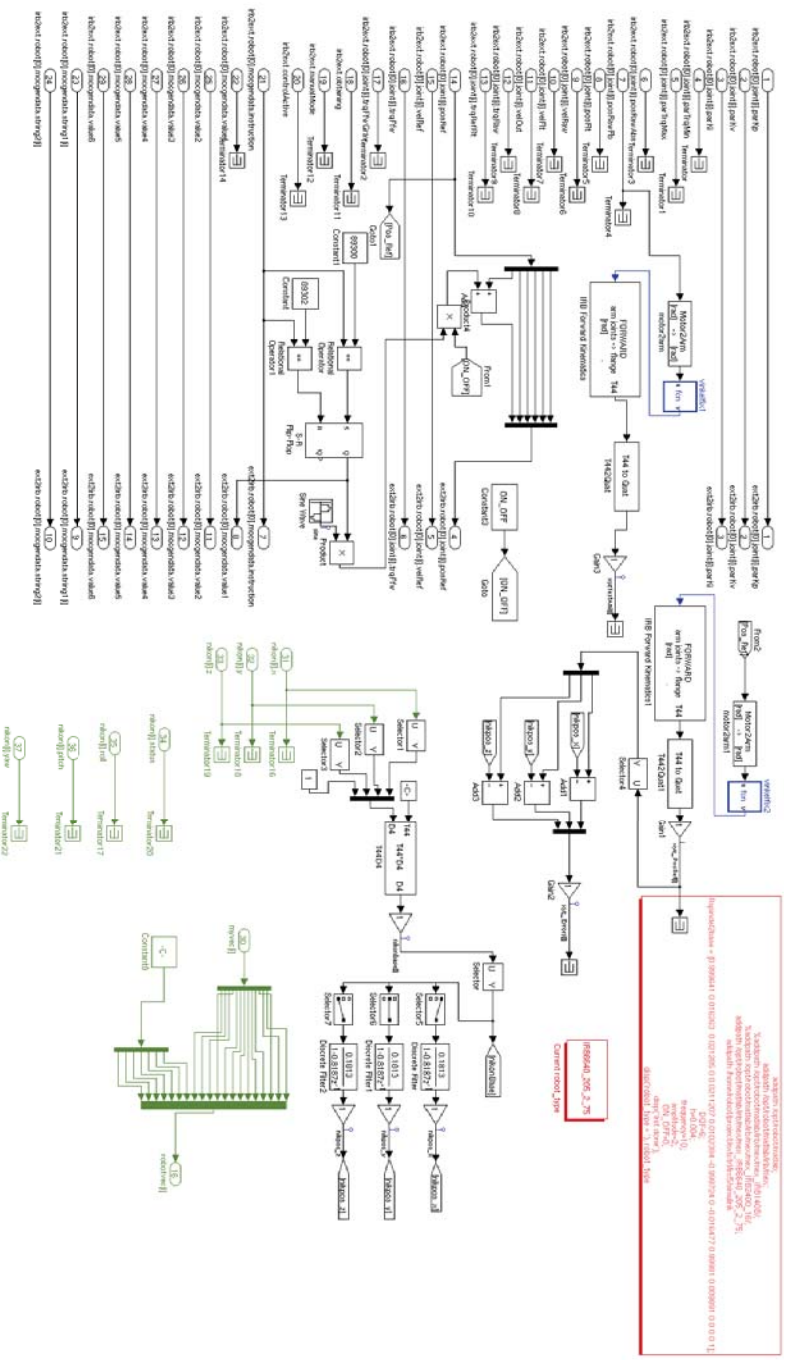
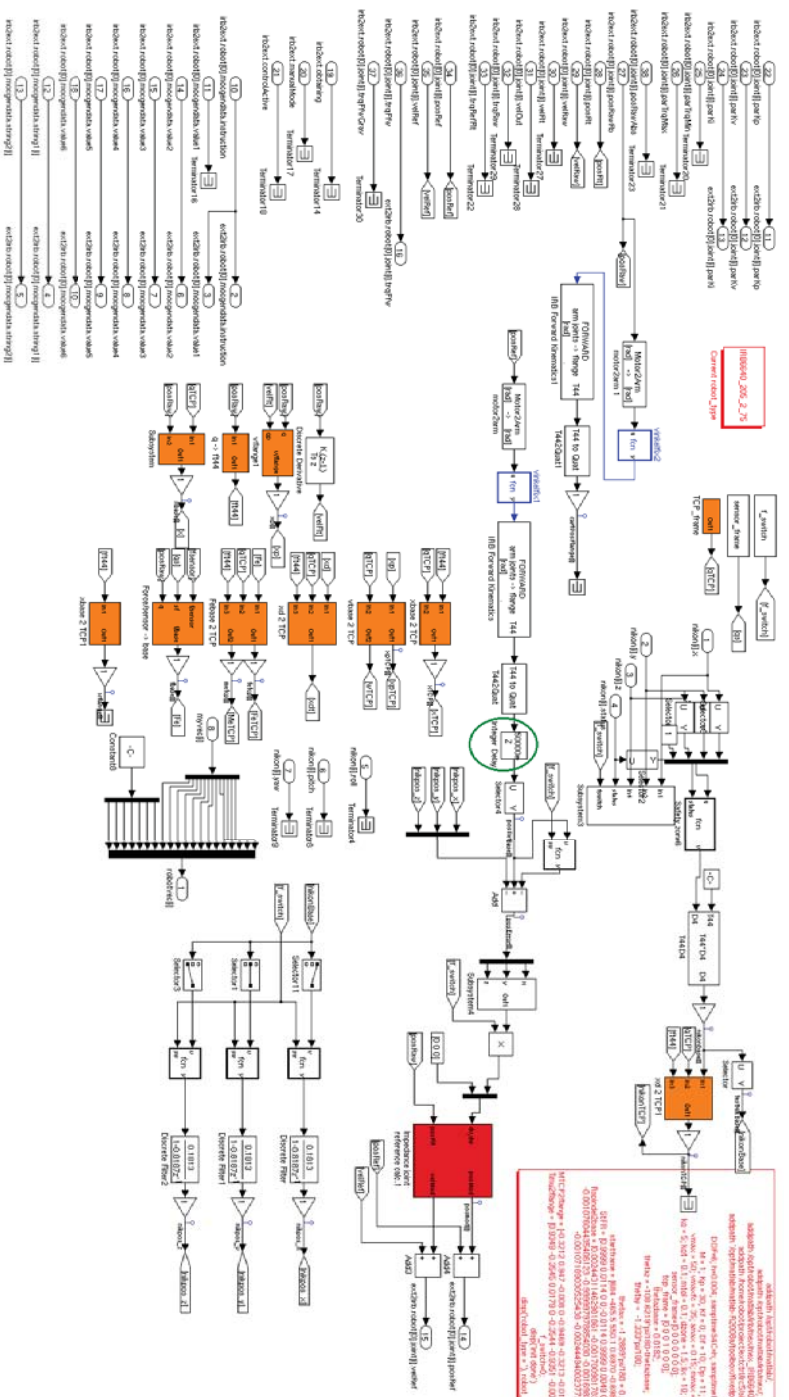


Figure A.1 Overview of the Simulink model used for kinematic verification.



100**Figure A.2** Overview of the Simulink model with the block *Integer Delay* (green oval).

B

RAPID Programs and Matlab Scripts

B.1 Original RapidBox Program

```
%%%
  VERSION:1
  LANGUAGE:ENGLISH
%%%

MODULE DELCAM_AML1
  PERS tooldata tdelcam1:=[FALSE,[[1985.957,-151.27,1165.804]
  ,[0.00021425,0.70447824,-0.00021585,0.70972552]],[5,[0,0,100]
  ,[1,0,0,0],0,0,0]];
  PERS wobjdata wdelcam1:=[TRUE,TRUE,"",[[0,-100,333.5]
  ,[0.70710678,0,0,0.70710678]],[[0,0,0],[1,0,0,0]]];

  PROC Delcam1_AML()
    ! Joint angles at start point : [!A1!,!A2!,!A3!,!A4!,!A5!,!A6!]
    ,[9E9,9E9,9E9,9E9,9E9,9E9]
    ConfJ\On;
    ConfL\Off;
    AccSet 5,100;
    WaitTime 2;
    MoveJ [[100.00,0.00,25.00],[0.707071,0,0,0.707142],[-1,-1,0,1]
    ,[9E9,9E9,9E9,9E9,9E9,9E9]],v20,z0,tdelcam1\WObj:=wdelcam1;
    MoveL [[100.00,0.00,25.00],[0.70708,8.94019E-06,1.40498E-05
    ,0.707134],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
    , v20, z0, tdelcam1\WObj:=wdelcam1;
    MoveL [[130.00,0.00,25.00],[0.707085,-1.91866E-05,5.24451E-05
    ,0.707129],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
```

```
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[130.00,-20.00,25.00],[0.707085,-1.82253E-05,5.12611E-05
,0.707128],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,-20.00,25.00],[0.707084,-1.95296E-05,5.34128E-05
,0.70713],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,0.00,25.00],[0.707085,-1.97948E-05,5.33866E-05
,0.707129],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,0.00,15.00],[0.707076,-1.40422E-05,6.3306E-05
,0.707138],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[130.00,0.00,15.00],[0.707088,-1.2471E-05,6.17851E-05
,0.707126],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[130.00,-20.00,15.00],[0.707087,-1.28744E-05,6.24542E-05
,0.707126],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,-20.00,15.00],[0.707088,-1.36733E-05,6.21325E-05
,0.707126],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,0.00,15.00],[0.707088,-9.23785E-06,5.8104E-05
,0.707125],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,0.00,5.00],[0.707089,-1.46001E-05,6.5498E-05
,0.707125],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[130.00,0.00,5.00],[0.707083,-1.33462E-05,5.95959E-05
,0.70713],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[130.00,-20.00,5.00],[0.707085,-1.57842E-05,5.97814E-05
,0.707129],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,-20.00,5.00],[0.707084,-1.41693E-05,6.02042E-05
,0.707129],[-1,-1,0,1]
,[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;
MoveL [[100.00,0.00,5.00],[0.707084,-1.03862E-05,5.6834E-05
,0.70713],[-1,-1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
, v20, z0, tdelcam1\WObj:=wdelcam1;

ConfJ\On;
```

```
    ConfL\On;
    ! Stop;
  ENDPROC
ENDMODULE
```

B.2 ILC Script

```
addpath /home/robot/project/extctrl/matlab/

% Read first nikpos-log from reference

%filename = '/work/thomas/Air_ILC_Mill_130531.txt'; % Air
filename = '/work/thomas/ILC_Mill_130531_P1.txt'; % Mill
[nikon,names]=readlog(filename,{'nikpos'});

%% Generate and filter signals from nikpos-log

% 1: Nikon data for "Air_ILC_Mill_130531.txt"
%indexStart= 952;
%indexStop= 9071;

% 2: Nikon data for "ILC_Mill_130531_P1.txt"
indexStart= 867;
indexStop= 8982;

xNikPosRaw = -nikon(indexStart:indexStop,2);
yNikPosRaw = -nikon(indexStart:indexStop,3);
zNikPosRaw = -nikon(indexStart:indexStop,1);

[b,a] = butter(6, 4/(0.5*125));

xNikPosBias = filtfilt(b, a, xNikPosRaw);
yNikPosBias = filtfilt(b, a, yNikPosRaw);
zNikPosBias = filtfilt(b, a, zNikPosRaw);

%% Calculate Start & End Points for Calculating Offset & Var

xPl = 5;
xMi = -5;

vSH1 = length(xyInOut) + length(x1P) +1;
vEH1 = vSH1 + length(x2P);
```

```
vSL1 = vEH1 + length(x3P);
vEL1 = vSL1 + length(x4P) + length(xyPause);

vSH2 = vEL1 + length(x1P);
vEH2 =  vSH2 + length(x2P);

vSL2 = vEH2 + length(x3P);
vEL2 = vSL2 + length(x4P) + length(xyPause);

vSH3 = vEL2 + length(x1P);
vEH3 = vSH3 + length(x2P);

vSL3 = vEH3 + length(x3P);
vEL3 = vSL3 + length(x4P) + length(xyInOut) -5;

%% Check X Start & End Points
set(0,'defaultlinelinewidth',2)

xF1 = ((vSH1+xPl):(vEH1+xMi)-5);
xF2 = ((vSL1+xPl):(vEL1+xMi)+5);
xF3 = ((vSH2):(vEH2+2*xMi));
xF4 = ((vSL2+xPl):(vEL2+xMi));
xF5 = ((vSH3):(vEH3+xMi));
xF6 = ((vSL3+xPl):(vEL3+xMi));

figure
plot(xNikPosBias, 'r')
hold on
plot(xF1, xNikPosBias(xF1), 'b')
hold on
plot(xF2, xNikPosBias(xF2), 'b')
hold on
plot(xF3, xNikPosBias(xF3), 'b')
hold on
plot(xF4, xNikPosBias(xF4), 'b')
hold on
plot(xF5, xNikPosBias(xF5), 'b')
hold on
plot(xF6, xNikPosBias(xF6), 'b')
%hold on
%plot(xPos)
grid on
```

```
%% Calculate X Offset

xOffMeanH1 = xLength -mean(xNikPosBias(xF1));
xOffMeanL1 = 0 -mean(xNikPosBias(xF2));

xOffMeanH2 = xLength -mean(xNikPosBias(xF3));
xOffMeanL2 = 0 -mean(xNikPosBias(xF4));

xOffMeanH3 = xLength -mean(xNikPosBias(xF5));
xOffMeanL3 = 0 -mean(xNikPosBias(xF6));

xOff = (xOffMeanH1 + xOffMeanL1 + xOffMeanH2 + xOffMeanL2
+ xOffMeanH3 + xOffMeanL3)/6

%% Check Y Start & End Points

yPl = 15;
yMi = -15;
ydX = length(xyPause);

yF1 = (yPl:(vSH1+yMi));
yF2 = ((vEH1+yPl):(vSL1+yMi));
yF3 = ((vEL1-ydX+yPl)+5:(vSH2+yMi));
yF4= ((vEH2+yPl)-5:(vSL2+yMi)-5);
yF5 = ((vEL2-ydX+yPl+10):(vSH3+yMi));
yF6 = ((vEH3+yPl)-10:(vSL3+yMi)-5);

figure
plot(yNikPosBias, 'r')
hold on
plot(yF1, yNikPosBias(yF1), 'b')
hold on
plot(yF2, yNikPosBias(yF2), 'b')
hold on
plot(yF3, yNikPosBias(yF3), 'b')
hold on
plot(yF4, yNikPosBias(yF4), 'b')
hold on
plot(yF5, yNikPosBias(yF5), 'b')
hold on
plot(yF6, yNikPosBias(yF6), 'b')
%hold on
```

```
%plot(xPos)
grid on

%% Calculate Y Offset

yOffMeanL1 = 0 - mean(yNikPosBias(yF1));
yOffMeanH1 = yLength -mean(yNikPosBias(yF2));

yOffMeanL2 = 0 - mean(yNikPosBias(yF3));
yOffMeanH2 = yLength -mean(yNikPosBias(yF4));

yOffMeanL3 = 0 - mean(yNikPosBias(yF5));
yOffMeanH3 = yLength -mean(yNikPosBias(yF6));

yOff = (yOffMeanL1 + yOffMeanH1 + yOffMeanL2 + yOffMeanH2
+ yOffMeanL3 + yOffMeanH3)/6

%% Check Z Start & End Points

zF1 = (yPl:(vEL1-ydX+yMi)-5);
zF2 = ((vEL1+yPl)-10:(vEL2-ydX+yMi)-10);
zF3 = ((vEL2+yPl)-10:(vEL3+yMi)-10);

figure
plot(zNikPosBias, 'r')
hold on
plot(zF1, zNikPosBias(zF1), 'b')
hold on
plot(zF2, zNikPosBias(zF2), 'b')
hold on
plot(zF3, zNikPosBias(zF3), 'b')
grid on

%% Calculate Z Offset

zOffMeanL1 = 0 -mean(zNikPosBias(zF1));
zOffMeanL2 = -1 -mean(zNikPosBias(zF2));
zOffMeanL3 = -2 -mean(zNikPosBias(zF3));

zOff = (zOffMeanL1 + zOffMeanL2 + zOffMeanL3)/3

% OFF X,Y,Z
Off = [xOff, yOff, zOff]
```

```
xNikPos = xNikPosBias +Off(1);
yNikPos = yNikPosBias +Off(2);
zNikPos = zNikPosBias +Off(3);

%% Calculate Var & Std for X,Y,Z

xH1 =  xLength -xNikPos(xF1);
xL1 = 0 -xNikPos(xF2);

xH2 =  xLength -xNikPos(xF3);
xL2 = 0 -xNikPos(xF4);

xH3 =  xLength -xNikPos(xF5);
xL3 = 0 -xNikPos(xF6);

xVec = [xH1' xL1' xH2' xL2' xH3' xL3'];
xVar = var(xVec);
xStdDer = std(xVec);

yL1 = 0 -yNikPos(yF1);
yH1 = yLength -yNikPos(yF2);

yL2 = 0 -yNikPos(yF3);
yH2 = yLength -yNikPos(yF4);

yL3 = 0 -yNikPos(yF5);
yH3 = yLength -yNikPos(yF6);

yVec = [yL1' yH1' yL2' yH2' yL3' yH3'];
yVar = var(yVec);
yStdDer = std(yVec);

zL1 = 0 -zNikPos(zF1);
zL2 = -1 -zNikPos(zF2);
zL3 = -2 -zNikPos(zF3);

zVec = [zL1' zL2' zL3'];
zVar = var(zVec);
zStdDer = std(zVec);

VarNoComp = [xVar yVar zVar]
StdDerNoComp = [xStdDer yStdDer zStdDer]
```

```
%% Plot shape/"toolpath" of reference and nikpos in X,Y

figure
plot(xPos, yPos, 'b')
hold on
plot(xNikPos, yNikPos, 'r')
%hold on
%plot(xILC, yILC, 'g')
xlabel('X Position [mm]')
ylabel('Y Position [mm]')
grid on

figure
plot(zPos, yPos, 'b')
hold on
plot(zNikPos, yNikPos, 'r')
xlabel('Z Position [mm]')
ylabel('Y Position [mm]')
grid on

%% Determine start and end point of nikpos-data

figure
plot(xNikPos, 'r')
%hold on
%plot(xPos)
grid on

figure
plot(yNikPos, 'r')
%hold on
%plot(yPos)
grid on

figure
plot(zNikPos, 'r')
%hold on
%plot(zPos)
grid on

%% Check filtered signal compared to unfiltered
```

```
figure
plot(yNikPosRaw)
hold on
plot(yNikPos, 'r')
grid on

%%
close all
%% Update Reference Parameters for Calculating ILC Offset & Var

xyInOutN2 = xyInOut;
x1PN2 = x1P;
x2PN2 = x2P;
x3PN2 = x3P;
x4PN2 = x4P;
xyPauseN2 = xyPause;

%% Determine step Value for ILC:1 Iteration
figure
plot(xPos, 'b')
%hold on
%plot(xNikPosBias+(0), 'r')
hold on
plot(xNewNikPosBias+(-56.61), 'r')
xlabel('X Position [mm]')
ylabel('Y Position [mm]')
grid on

%% Generate Reference

xLength= 32;
yLength= -22;

% AIR:
%step= (1/24.82); % Original;
%% 0 Iteration "Air_ILC_Mill_130531.txt".
%step= (1/24.96); % PV2:Pos:
%% 1 Iteration: compAir_ILC_Mill_130603.txt".

% MILL:
%step= (1/(24.82)); % P1: Mill Original;
%% 0 Iteration "ILC_Mill_130531_P1.txt".
%step= (1/25.0); % P2: PV2-Pos: 1
```

```
%% 1 Iteration: P2: compMill_ILC_Mill_130603_P2.txt".
%step= (1/24.95); % P5: PV2-Pos: 1
%% 1 Iteration: P5: airCompV10_ILC_Mill_130603_P5.txt".
%step= (1/(49.97)); % P6: PV2-Pos: 1
%% 1 Iteration: P6: compMill_V5_ILC_Mill_130603_P6.txt".
%step= (1/(49.61)); % P4: PV2-Pos: 1
%% 1 Iteration: P4: airCompV5_ILC_Mill_130603_P4.txt".

%step= (1/(26)); % Low Value Demo".
%step= (1/(24.84)); % High Value Demo".

x1P = 0: step: xLength;
y1P = zeros(1,length(x1P));

y2P = 0: -step: yLength;
x2P = xLength*ones(1, length(y2P));

% []

x3P = xLength: -step: 0;
y3P = yLength*ones(1, length(x3P));

y4P = yLength: step: 0;
x4P = zeros(1, length(y4P));

xBoxPos = [x1P x2P x3P x4P];
yBoxPos = [y1P y2P y3P y4P];

zPause1= zeros(1, length(yBoxPos));
z1P= 0: -step: -1;
zPause2= -1*ones(1, length(yBoxPos));
z2P= -1: -step: -2;
zPause3= -2*ones(1, length(yBoxPos));

xyInOut = zeros(1,10);
zIn = xyInOut;
zOut = -2*ones(1,10);
xyPause = zeros(1,length(z1P));

zPos = ([zIn zPause1 z1P zPause2 z2P zPause3 zOut])';

xPos = ([xyInOut xBoxPos xyPause xBoxPos xyPause xBoxPos xyInOut])';
yPos = ([xyInOut yBoxPos xyPause yBoxPos xyPause yBoxPos xyInOut])';
```

```
%% Test of timing in RAPID-program --> all synced
figure
plot(-xPos, 'b')
hold on
plot(-yPos, 'r')
hold on
plot(zPos, 'g')
grid on

%% Generate Reference RAPID Code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

box_file = fopen('box_file.txt','w');

str = '';
for i = 1:2:length(xPos)
    str = [str, '\t', 'MoveL [[', num2str(xPos(i) +83.0, '%0.5f'),
    ',* ', num2str(yPos(i) +34.5, '%0.5f'),
    ',* ', num2str(zPos(i) +46.0, '%0.5f'), '],[0.707139,0.0,0.0,
    0.707075],[-1,-1,0,1],* [9E9,9E9,9E9,
    9E9,9E9,9E9]], v10, z0, tdelcam1\\WObj:=wdelcam1;', '\n'];
    %sprintf('%0.2f',str);
end

fprintf(box_file, str);

fclose(box_file);

%%
close all

%% Generate ILC-reference from Errors in x,y,z

for i = 1:length(xPos)
    xError(i) = xPos(i) - xNikPos(i);
    xILC(i) = xPos(i) +xError(i);
    yError(i) = yPos(i) - yNikPos(i);
    yILC(i) = yPos(i) +yError(i);
    zError(i) = zPos(i) - zNikPos(i);
    zILC(i) = zPos(i) +zError(i);
end
```

```
figure
plot(xError, 'g')
hold on
plot(xPos, 'k')
hold on
plot(xNikPos, 'r')
hold on
plot(xILC, 'b')
grid on

figure
plot(yError, 'g')
hold on
plot(yPos, 'k')
hold on
plot(yNikPos, 'r')
hold on
plot(yILC, 'b')
grid on

figure
plot(zError, 'g')
hold on
plot(zPos, 'k')
hold on
plot(zNikPos, 'r')
hold on
plot(zILC, 'b')
grid on

%%
close all

%% ILC RAPID CODE GENERATION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate ILC Iteration:1; RAPID Code

box_file = fopen('box_file.txt','w');

str = '';
for i = 1:2:length(xILC)
    str = [str, '\t', 'MoveL [[', num2str(xILC(i) +83.0, '%0.5f'),
```

```
     ',* ', num2str(yILC(i) +34.5, '%0.5f'),
    ',* ', num2str(zILC(i) +46.0, '%0.5f'), '],[0.707139,0.0,0.0,
    0.707075],[-1,-1,0,1],* [9E9,9E9,9E9,
    9E9,9E9,9E9]], v10, z0, tdelcam1\\WObj:=wdelcam1;', '\n'];
    %sprintf('%0.2f',str);
end

fprintf(box_file, str);

fclose(box_file);

%% READ ILC:1:iter LOG txt-file

% Read ILC:1 First Iteration nikpos-log

%newFilename = '/work/thomas/compAir_ILC_Mill_130603.txt';
newFilename = '/work/thomas/compMill_ILC_Mill_130603_P2.txt';
%newFilename =
'/work/thomas/compMill_V5_ILC_Mill_130603_P6.txt';
[newNikon,names]=readlog(newFilename,{'nikpos'});

%% CALCULATE ILC:1 SIGNALS

% Nikon Data ILC-Air: compAir_ILC_Mill_130603.txt';
%newStart = 1095;
%newStop = 9259;

% Nikon Data P2: compMill_ILC_Mill_130603_P2.txt';
newStart = 1047;
newStop = 9224;

% Nikon Data P6: compMill_V5_ILC_Mill_130603_P6.txt'
%newStart = 1273;
%newStop = 17590;

xNewNikPosRaw = -newNikon(newStart:newStop,2);
yNewNikPosRaw = -newNikon(newStart:newStop,3);
zNewNikPosRaw = -newNikon(newStart:newStop,1);

[b,a] = butter(6, 4/(0.5*125));

xNewNikPosBias = filtfilt(b, a, xNewNikPosRaw);
yNewNikPosBias = filtfilt(b, a, yNewNikPosRaw);
```

```
zNewNikPosBias = filtfilt(b, a, zNewNikPosRaw);

%% Calculate ILC:1 Start & End Points for Calculating Offset & Var
xNPl = 0;
xNMi = -10;

vNSH1 = length(xyInOutN2) + length(x1PN2) +1;
vNEH1 = vNSH1 + length(x2PN2);

vNSL1 = vNEH1 + length(x3PN2);
vNEL1 = vNSL1 + length(x4PN2) + length(xyPauseN2);

vNSH2 = vNEL1 + length(x1PN2);
vNEH2 =  vNSH2 + length(x2PN2);

vNSL2 = vNEH2 + length(x3PN2);
vNEL2 = vNSL2 + length(x4PN2) + length(xyPauseN2);

vNSH3 = vNEL2 + length(x1PN2);
vNEH3 = vNSH3 + length(x2PN2);

vNSL3 = vNEH3 + length(x3PN2);
vNEL3 = vNSL3 + length(x4PN2) + length(xyInOutN2) -5;

%% Check X Start & End Points ILC

xNF1 = ((vNSH1+xNPl):(vNEH1+xNMi)-10);
xNF2 = ((vNSL1+xNPl)-10:(vNEL1+xNMi)+10);
xNF3 = ((vNSH2+xNPl)+5:(vNEH2+xNMi));
xNF4 = ((vNSL2+xNPl)-5:(vNEL2+xNMi));
xNF5 = ((vNSH3+xNPl):(vNEH3+xNMi));
xNF6 = ((vNSL3+xNPl):(vNEL3+xNMi));

figure
plot(xNewNikPosBias, 'r')
hold on
plot(xNF1, xNewNikPosBias(xNF1), 'b')
hold on
plot(xNF2, xNewNikPosBias(xNF2), 'b')
hold on
plot(xNF3, xNewNikPosBias(xNF3), 'b')
hold on
plot(xNF4, xNewNikPosBias(xNF4), 'b')
```

```
hold on
plot(xNF5, xNewNikPosBias(xNF5), 'b')
hold on
plot(xNF6, xNewNikPosBias(xNF6), 'b')
%hold on
%plot(xPos)
xlabel('Point Index')
ylabel('X Position [mm]')
grid on

%% Calculate X Offset ILC

xNewOffMeanH1 = xLength -mean(xNewNikPosBias(xNF1));
xNewOffMeanL1 = 0 -mean(xNewNikPosBias(xNF2));

xNewOffMeanH2 = xLength -mean(xNewNikPosBias(xNF3));
xNewOffMeanL2 = 0 -mean(xNewNikPosBias(xNF4));

xNewOffMeanH3 = xLength -mean(xNewNikPosBias(xNF5));
xNewOffMeanL3 = 0 -mean(xNewNikPosBias(xNF6));

xNewOff = (xNewOffMeanH1 + xNewOffMeanL1 + xNewOffMeanH2
+ xNewOffMeanL2 + xNewOffMeanH3 + xNewOffMeanL3)/6

%% Check Y Start & End Points ILC

yNPl = 10;
yNMi = -15;
yNdx = length(xyPauseN2);

yNF1 = (yNPl:(vNSH1+yNMi));
yNF2 = ((vNEH1+yNPl)-5:(vNSL1+yNMi))-5;
yNF3 = ((vNEL1-yNdx+yNPl):(vNSH2+yNMi));
yNF4= ((vNEH2):(vNSL2+yNMi)-5);
yNF5 = ((vNEL2-yNdx+yNPl)+10:(vNSH3+yNMi))-15;
yNF6 = ((vNEH3+yNPl)-5:(vNSL3+yNMi)-5);

figure
plot(yNewNikPosBias, 'r')
hold on
plot(yNF1, yNewNikPosBias(yNF1), 'b')
hold on
plot(yNF2, yNewNikPosBias(yNF2), 'b')
```

```
hold on
plot(yNF3, yNewNikPosBias(yNF3), 'b')
hold on
plot(yNF4, yNewNikPosBias(yNF4), 'b')
hold on
plot(yNF5, yNewNikPosBias(yNF5), 'b')
hold on
plot(yNF6, yNewNikPosBias(yNF6), 'b')
%hold on
%plot(xPos)
xlabel('Point Index')
ylabel('Y Position [mm]')
grid on

%% Calculate Y Offset ILC

yNewOffMeanL1 = 0 - mean(yNewNikPosBias(yNF1));
yNewOffMeanH1 = yLength -mean(yNewNikPosBias(yNF2));

yNewOffMeanL2 = 0 - mean(yNewNikPosBias(yNF3));
yNewOffMeanH2 = yLength -mean(yNewNikPosBias(yNF4));

yNewOffMeanL3 = 0 - mean(yNewNikPosBias(yNF5));
yNewOffMeanH3 = yLength -mean(yNewNikPosBias(yNF6));

yNewOff = (yNewOffMeanL1 + yNewOffMeanH1 + yNewOffMeanL2
+ yNewOffMeanH2 + yNewOffMeanL3 + yNewOffMeanH3)/6

%% Check Z Start & End Points ILC

zNF1 = (yNPl:(vNEL1-yNdx+yNMi)+5);
zNF2 = ((vNEL1+yNPl)+10:(vNEL2-yNdx+yNMi)-5);
zNF3 = ((vNEL2+yNPl):(vNEL3+yNMi));

figure
plot(zNewNikPosBias, 'r')
hold on
plot(zNF1, zNewNikPosBias(zNF1), 'b')
hold on
plot(zNF2, zNewNikPosBias(zNF2), 'b')
hold on
plot(zNF3, zNewNikPosBias(zNF3), 'b')
xlabel('Point Index')
```

```
ylabel('Z Position [mm]')
grid on

%% Calculate Z Offset ILC

zNewOffMeanL1 = 0 -mean(zNewNikPosBias(zNF1));
zNewOffMeanL2 = -1 -mean(zNewNikPosBias(zNF2));
zNewOffMeanL3 = -2 -mean(zNewNikPosBias(zNF3));

zNewOff = (zNewOffMeanL1 + zNewOffMeanL2 + zNewOffMeanL3)/3

% NewOff X,Y,Z
NewOff = [xNewOff, yNewOff, zNewOff]

xNewNikPos = xNewNikPosBias +NewOff(1);
yNewNikPos = yNewNikPosBias +NewOff(2);
zNewNikPos = zNewNikPosBias +NewOff(3);

%%  Calculate ILC Var & Std for X,Y,Z

xNH1 =  xLength -xNewNikPos(xNF1);
xNL1 = 0 -xNewNikPos(xNF2);

xNH2 =  xLength -xNewNikPos(xNF3);
xNL2 = 0 -xNewNikPos(xNF4);

xNH3 =  xLength -xNewNikPos(xNF5);
xNL3 = 0 -xNewNikPos(xNF6);

xNewVec = [xNH1' xNL1' xNH2' xNL2' xNH3' xNL3'];
xNewVar = var(xNewVec);
xNewStdDer = std(xNewVec);

yNL1 = 0 -yNewNikPos(yNF1);
yNH1 = yLength -yNewNikPos(yNF2);

yNL2 = 0 -yNewNikPos(yNF3);
yNH2 = yLength -yNewNikPos(yNF4);

yNL3 = 0 -yNewNikPos(yNF5);
yNH3 = yLength -yNewNikPos(yNF6);

yNewVec = [yNL1' yNH1' yNL2' yNH2' yNL3' yNH3'];
```

```
yNewVar = var(yNewVec);
yNewStdDer = std(yNewVec);

zNL1 = 0 -zNewNikPos(zNF1);
zNL2 = -1 -zNewNikPos(zNF2);
zNL3 = -2 -zNewNikPos(zNF3);

zNewVec = [zNL1' zNL2' zNL3'];
zNewVar = var(zNewVec);
zNewStdDer = std(zNewVec);

VarILC = [xNewVar yNewVar zNewVar]
StdDerILC = [xNewStdDer yNewStdDer zNewStdDer]

%%  PRINT: Var & Std (Before (ILC:0) & After (ILC:1))

VarNoComp
VarILC
StdDerNoComp
StdDerILC

% IMPROVEMENT in Std With ILC

xImproveStdILC = 1 - xNewStdDer/xStdDer;
yImproveStdILC = 1 - yNewStdDer/yStdDer;
zImproveStdILC = 1 - zNewStdDer/zStdDer;

ImproveStdILC = [xImproveStdILC yImproveStdILC zImproveStdILC]

%% ILC:1: Determine start and end point of nikpos-data
figure
plot(xPos, 'r')
hold on
plot(xNewNikPosRaw, 'b')
grid on

figure
plot(yNewNikPosRaw)
hold on
plot(yPos, 'r')
grid on

figure
```

```
plot(zNewNikPosRaw)
hold on
plot(zPos, 'r')
grid on

%% X,Y: ILC:1: Plot shape/"toolpath" of reference and nikpos

figure
plot(xPos, yPos, 'b--')
hold on
plot(pXpath, pYpath, 'Color',grey);
hold on
plot(pXtool, pYtool, 'Color',grey);
hold on
plot(xNikPos, yNikPos, 'r')
hold on
plot(xNewNikPos, yNewNikPos, 'g')
xlabel('X Position [mm]')
ylabel('Y Position [mm]')
grid on

%% Y,X: ILC:1: Plot shape/"toolpath" of reference and nikpos

figure
plot(yPos, xPos, 'b--')
hold on
plot(pYpath, pXpath, 'Color',grey)
hold on
plot(pYtool, pXtool, 'Color',grey)
hold on
plot(yNikPos, xNikPos, 'r')
hold on
plot(yNewNikPos, xNewNikPos, 'g')
xlabel('Y Position [mm]')
ylabel('X Position [mm]')
grid on

%%  Z,Y: ILC:1: Plot shape/"toolpath" of reference and nikpos

figure
plot(pZVpath, pYVpath, 'Color',grey);
hold on
plot(pZVtool, pYVtool, 'Color',grey);
```

```
plot(zPos, yPos, 'b--')
hold on
plot(zNikPos, yNikPos, 'r')
hold on
plot(zNewNikPos, yNewNikPos, 'g')
xlabel('Z Position [mm]')
ylabel('Y Position [mm]')
grid on

%% Z Index and Err Lim

zLimH = 0.1*ones(1,length(zPos));
zLimL = -0.1*ones(1,length(zPos));

figure
plot(zLimH, 'Color',grey);
hold on
plot(zLimL, 'Color',grey);
plot(zPos, 'b--')
hold on
plot(zNikPos, 'r')
hold on
plot(zNewNikPos, 'g')
xlabel('Point Index')
ylabel('Z Position [mm]')
grid on
```

Bibliography

ABB Robotics (2004). *The new IRC5 robot controller*. ABB AB. Västerås, Sweden.

— (2007). *IRB 6640 — A stronger robot - the next generation*. ABB AB. Västerås, Sweden.

— (2010). *IRB 6640 — Product specification, Articulated robot*. 3HAC 028284-001. ABB AB. Västerås, Sweden.

— (2011). *IRC5 — Industrial Robot Controller*. ABB AB. Västerås, Sweden.

Arimoto, S., S. Kawamura, and F. Miyazaki (1984). "Bettering operation of robots by learning". *Journal of Robotic systems* **1**:2.

Berselli, D., O. Bailey, B. Leach, M. Dunning, R. Krain, C. Lehmann, J. Van der Zwaag, and A. Kerstens (2013a). *COMET Deliverable 5.1 — Case study: Aerospace components*. Tech. rep. COMET.

Berselli, D., F. Leali, F. Pini, C. Lehmann, J. Van der Zwaag, and A. Kerstens (2013b). *COMET Deliverable 5.2 — Case study: Automotive*. Tech. rep. COMET.

Blomdell, A., I. Dressler, K. Nilsson, and A. Robertsson (2010). "Flexible Application Development and High-performance Motion Control Based on External Sensing and Reconfiguration of ABB Industrial Robot Controllers". In: *Proc. ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*. Anchorage, AK.

COMET (2013). `www.cometproject.eu`. COMET — Plug-and-produce COmponents and METhods for adaptive control of industrial robots enabling cost effective, high precision manufacturing in factories of the future.

Dressler, I. (2009). "Force control interface for ABB S4/IRC5". Upublished Document.

— (2012). *Modeling and Control of Stiff Robots for Flexible Manufacturing*. PhD thesis ISRN LUTFD2/TFRT--1093--SE. Department of Automatic Control, Lund University, Sweden.

Hakvoort, W., R. Aarts, J van Dijk, and J. Jonker (2007). "Model-based iterative learning control applied to an industrial robot with elasticity". In: *Decision and Control, 2007 46th IEEE Conference on*. IEEE, pp. 4185–4190.

KDE (2013). *Kst — Visualize your data*. http://kst-plot.kde.org.

Nikon Metrology (2010). *K-Series Optical CMM Solutions — Supporting a variety of metrology applications*. Nikon.

— (2013). *K-Series Optical CMM*. `www.nikonmetrology.com/en_EU/Products/Portable-Measuring/Optical-CMM/K-Series-Optical-CMM/(key_features)`. Nikon.

Norrlöf, M. (2000). *Iterative Learning Control — Analysis, Design and Experiments*. No. 653. PhD thesis. Department of Electrical Engineering, Linköpings Universitet, Sweden.

Olofsson, B. (2013). *Topics in Machining with Industrial Robots and Optimal Control of Vehicles*. Licentiate Thesis ISRN LUTFD2/TFRT--3259--SE. Department of Automatic Control, Lund University, Sweden.

Spong, M. W. and M. Vidyasagar (1989). *Robot Dynamics and Control*. John Wiley and Sons, New York.

Uchiyama, M. (1978). "Formulation of high-speed motion pattern of a mechanical arm by trial". *Trans. SICE (Soc. Instrum. Contr. Eng.)* **14**:6. Published in Japanese.

Wang, J., H. Zhang, and T. Fuhlbrigge (2009). "Improving machining accuracy with robot deformation compensation". In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, pp. 3826–3831.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
May 2014

Document Number
ISRN LUTFD2/TFRT--5942--SE

Author(s)
Thomas Daun

Supervisor
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Björn Olofsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle

Iterative Learning Control for Milling with Industrial Robots in Advanced Manufacturing

Abstract

The demand of today for advanced manufactured parts with high precision is due to the increasing complexity of technologies. The parts are typically made by CNC (Computer Numerical Control) machines, which are expensive and comparably big. By using industrial robots that are significantly cheaper, reduced costs can be achieved, which is particularly beneficial for small and medium enterprises. Robots are, however, less stiff and strong and are less accurate compared to the CNC machines.

In this thesis, the idea was that by designing a controller, this could be compensated for so that the robot could perform machining with high precision. The research made in this thesis was part of the EU co-funded research project COMET.

The robot task was to repeatedly mill parts with improved results. Iterative Learning Control (ILC) was therefore chosen as a suitable control strategy for this task. It uses the position error from previous iterations and adds it to the control signal to converge the output to successful results. Results showed that when using ILC for tracking paths, the position error could be reduced with approximately 11-20% in x, y, and z directions after one iteration.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-122

Recipient's notes

Security classification