Real-time quintic Hermite interpolation for robot trajectory execution

Morten Lind[*]

SINTEF Manufacturing, Trondheim, Norway

ABSTRACT

This paper presents a real-time joint trajectory interpolation system for the purpose of frequency scaling the low cycle time of a robot controller, allowing a Python application to real-time control the robot at a moderate cycle time. Interpolation is based on quintic Hermite piece-wise splines. The splines are calculated in real-time, in a piecewise manner between the high-level, long cycle time trajectory points, while sampling of these splines at an appropriate, shorter cycle time for the real-time requirement of the lower-level system. The principle is usable in general, and the specific implementation presented is for control of the Panda robot from Franka Emika. Tracking delay analysis is presented based on a cosine trajectory. A simple test application has been implemented, demonstrating real-time feeding of a pre-calculated trajectory for cutting with a knife. Estimated forces on the robot wrist are recorded during cutting and presented in the paper.

INTRODUCTION

Reacting instantaneously to perceived changes in the environment is an important feature of modern robotics, recently for safe human-robot cooperation, but also more traditionally for force-based control and visual servoing applications. The setup for this includes a sensor system for perceiving the environment, a control model for deciding how to react, and a control interface to the mechanical system. This must all be real-time integrated into a sensor-based robot control application.

A real-time interface to the robot arm, through the robot controller, is a necessity. Newer robot controllers are getting increasingly open towards real-time trajectory feeding. An early example of a supported mechanism was realized in the KUKA controllers with the Robot Sensor Interface (RSI) software package, which used a cycle time of 12 ms in version 2 and 4 ms in version 3; version 3 maintained the legacy 12 ms cycle time as an option. Results were presented by Lind et al. (2010).

The Universal Robots controllers can be controlled in various, official ways through the standard controller process. Methods and results are reported by Andersen (2015). Older versions of the Universal Robots controllers provided an unofficial C API, with very low tracking delay, as presented by Lind et al. (2010). More accurate methods for measuring and classifying types of delays was presented by Andersen et al. (2015).

Besides standard interfaces to robot controllers, some controllers can be hacked for gaining access at the lower controller levels. Kröger and Wahl (2010) directly accessed the frequency inverters of a Stäubli RX60 controller and was able to perform servo-level control in 10 kHz. Schrimpf (2013) injected a single-board computer in the USB-link between the higher and lower level controller components in a Nachi AX10 controller, enabling joint-level control in 100 Hz.

Developing sensor-based, real-time robot control applications is challenging. More so when the target programming platforms, in general, are exclusively available in compiled languages such as C++. For those situation where the real-time performance is not too critical, also called soft real-time requirements, it will increase the development efficiency, if a higher level, general programming language, such as Python, can work as the target platform. The main motivation for the presented work is to enable such a high level, general, soft real-time programming platform, in this case Python, for developing advanced control application; such as the DeepMPC framework for cutting of food products described by Lenz et al. (2015) and the real-time grasping control described by Morrison et al. (2018).

[*]morten.lind@sintef.no

From experiments and experience, a pure Python-based framework, exploiting NumPy for numerical calculations, manages to calculate kinematics for a typical industrial robot system with six to nine joints in 10 ms, with room for also doing some sensor processing and general data accounting, using a contemporary PC (Lind et al., 2012). This calculation time strongly depends on the amount for sensor processing and data accounting, and on the computing power on the CPU and computer system on which the software is deployed. On a modern, high-end PC of today the results presented by Lind et al. (2012) may well be possible with a 5 ms cycle time. Due to the "Global Interpreter Lock" in the Python interpreter, multi-threading in a single Python process is not utilizing the multiple cores of modern CPUs, and computationally heavy applications should be distributed over several processes. Eggen and Eggen (2019) presents experiments and results aimed at determining when it is advantageous to process-distribute a Python application. Schrimpf et al. (2013) presented a time analysis for various data paths in a distributed, real-time, sensor-based robot application implemented and deployed with Python and ROS.

The seven-axis Panda robot from Franka Emika can be controlled in real-time through an Ethernet UDP connection. The real-time control interface requires a cycle time of 1 ms. Several other robot controllers provide real-time interfaces for trajectory feeding at rates higher than 100 Hz. Mihelj et al. (2012) mentions official and research interfaces for real-time control through available controllers for KUKA, Stäubli, Yaskawa Motoman, and Comau robots. These examples may indicate a slow but general tendency for such joint-level, low-latency interfaces to become of more general availability among robot controllers in the future.

The cycle time requirement of 1 ms of the Franka controller, and even the 4 ms interface to newer KUKA controllers with RSI version 3, poses a computational and real-time challenge for Python-based real-time trajectory feeding application. It is a computation time challenge, since *pure* Python computations in general is slower than C++ code by an order of magnitude or more. This, however, is mitigated somewhat when using efficient scientific computation packages such as NumPy and SciPy. It is a real-time challenge, since the wake-up latency on the PC gets an additional contribution through the Python interpreter.

This paper shortly sketches a prototype solution addressing these challenges by presenting a joint-level intermediary motion service over Ethernet that respects the 1 ms real-time obligation towards the Franka controller, while providing an interface to the Python motion application requiring a more moderate cycle time; e.g. 10 ms. The motion service requires a cyclic real-time response from the motion application while obeying the similar requirement from the Franka controller. This allows for the motion application to make instantaneous changes and correction to the generated trajectory it emits. A positive side effect of this motion service is that it is re-connectable, since it detects the disconnection from, or a failure to comply in, the Python application, whereby it takes over the real-time obligation towards the Franka controller; and thus the low-level control loop is maintained.

The implementation of the presented solution is based on piece-wise quintic Hermite splines with an option to limit velocity, acceleration and jerk. The current implementation was developed for being fed a position trajectory from the motion application. However, velocities and accelerations are derived over the position trajectory and used in the interpolation, and can also be fed from the motion application. The presented solution can be classified as Type V in the scheme by Kröger and Wahl (2010).

The choice of piece-wise Hermite spline as the fundamental computational object for trajectory representation was made due to its adequate parameterization and domain: Piece-wise Hermite splines are parameterized by clamping the direct motion quantities, such as position, velocity, acceleration, and jerk, at the end-points of each segment.

SYSTEM AND METHODS

The minimal system that have been set up for experimentation and testing is based on the seven-axis Panda robot from Franka Emika[1]. An intermediate motion service is performing the interpolation from a lower frequency loop to the required rate of the high frequency loop for positions, and optionally velocities and accelerations, for all joints. A motion application may then connect to the low-frequency interface of this motion service, and thus control the robot arm in a moderate real-time frequency. This section gives an overview of the active entities in the experiment system and the Hermite spline calculus.

---

[1]https://franka.de/

(a) Rough structure of processes and entities with their connections. [Image credit: Morten Lind, 2020]

(b) Sequence diagram for the synchronization among the three threads in MS, the FA and the MA. [Image credit: Morten Lind, 2020]

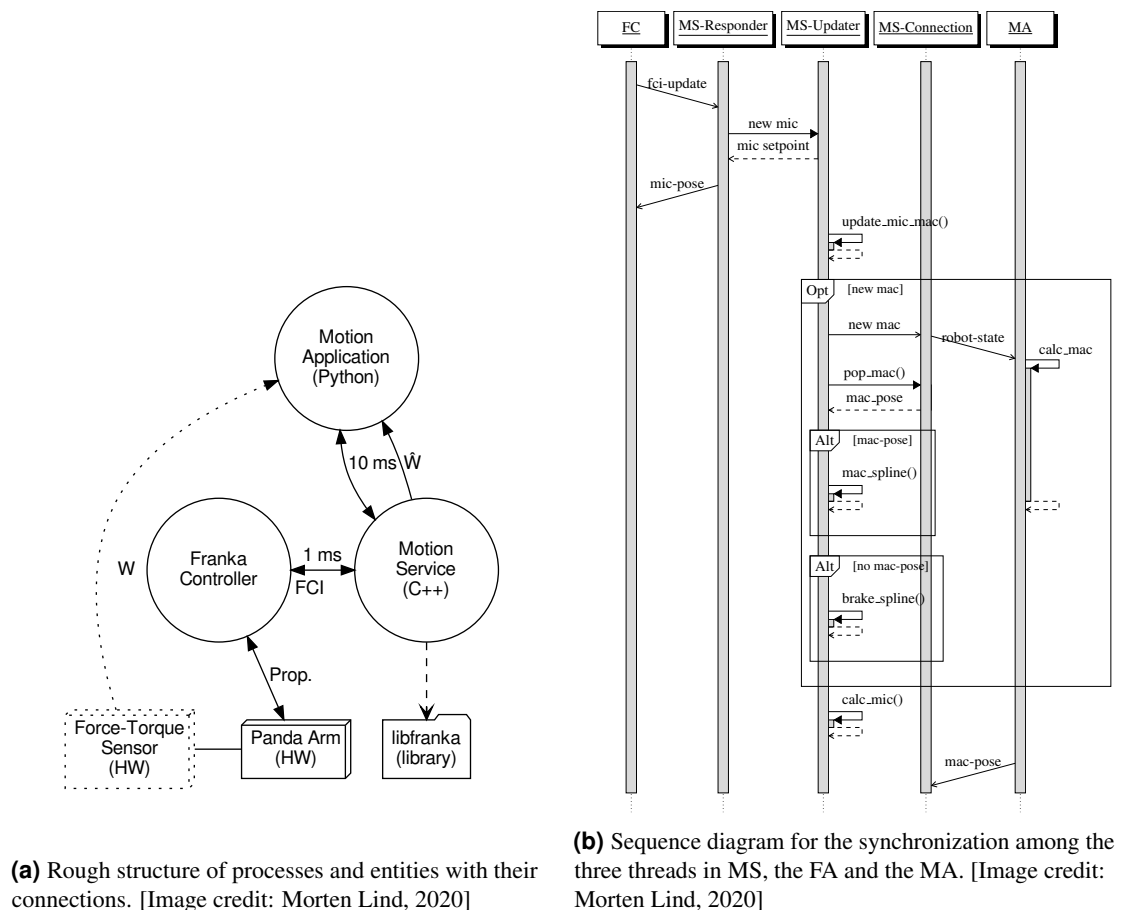



Figure 1. Interaction and sequence diagrams for the proposed system.

The Panda Robot and the Franka Controller

At the lowest level the robot arm servos are controlled via a proprietary interface by the "Franka Controller" (FC). The "Franka Control Interface"[2] (FCI) is a software addon for the FC which enables real-time trajectory feeding, synchronized with reading the state of the robot; including joint torques and estimated external forces.

When the FCI is installed on the FC, the freely available C++ library "libfranka"[3] can be used to operate the robot arm at a cycle time of 1 ms. The mechanism is to hand a callback function to libfranka when starting the control loop. This callback is then invoked every 1 ms with the status of the robot arm, expecting to be returned within less than one cycle. This cycle time is dubbed the *micro* cycle time.

In Fig. 1a FC is modelled as a process controlling the Panda arm, running on the Franka Controller PC, i.e. it is born with its own node. The FCI is exposed over an Ethernet UDP socket, on an IP address configured in the FC. Thus the trajectory feeding is performed from a different node; e.g. a user-supplied PC. libfranka provides various methods of control, ranging from operation (Cartesian) space control of the tool flange to joint torque control. This paper does not go into detail with libfranka and the presented work only uses the joint position control mode.

Motion Service and Application

For the purpose of obeying the micro cycle real-time requirement the "Motion Service" (MS) has been developed and implemented. As illustrated in Fig. 1a, MS links with libfranka for setting up the callback communicating with the FC over FCI, and starting the control loop.

The MS is running in its own process, on its own node, separate from the FC. The "Motion Application"

---

[2]https://frankaemika.github.io/docs/libfranka.html
[3]https://github.com/frankaemika/libfranka

117 (MA) process may run on the same computational node as MS. For connection responsiveness, it is better
118 to have the MS and the MA run on the same node. However, for dedicating more computing power to the
119 MA, leaving more computational resources for sensor analysis and control algorithms, it could be running
120 on a dedicated node. The presented architecture, where the MS is exposing its interface via an Ethernet
121 UDP socket, is flexible in this regard.

122   Figure 1a illustrates how an optional force-torque sensor may be added. Such a force-torque sensor
123 would typically be mechanically mounted between the wrist of the robot arm and the robot tool for
124 measuring the wrench, W, on the tool. A force-torque sensor is not used in the presented setup and
125 experiments. Instead the estimated wrench, $\hat{W}$ from the robot dynamics is used. This estimate is obtained
126 as a feature from libfranka and provided by the MS to the MA.

Software Mechanisms

128 For understanding the general workings of the MS, this section gives a brief overview of the central
129 mechanistic design.

130   Figure 1b shows a sequence diagram illustrating the central, operative synchronization and respon-
131 sibility of the three threads of the MS code. In addition the network-remote entities FC and MA are
132 modelled as threads with remote message interaction to the MS threads.

133   In addition to this central, operative interaction the connection thread in the MS has a complex logic
134 for maintaining control of the connection from an MA. This will not be treated here. Even though the
135 connection management, which enables the FC-MS interaction to be kept alive over repeated MA-sessions,
136 the main focus of this paper is to describe the interpolation in the MS.

137   Also not illustrated in Fig. 1b is the setup and initialization procedures of the MS, which establishes
138 the real-time loop with the FC and prepares for receiving connections from an MA. Like the connection
139 maintenance, this is also important, but the details are not in the focus of this paper.

140   A micro cycle is initiated by the callback from the FC to the Responder thread in the MS, conveying
141 the robot arm status. The Responder thread is only responsible for conveying this trigger and status data
142 to the Updater thread, and for returning the prepared next micro setpoint to the FC obtained from the
143 Updater thread in response. Upon receiving the new micro state and returning the next micro setpoint,
144 Updater thread is triggering itself to calculate the micro setpoint for the next micro cycle.

145   When the Updater thread is triggered to prepare the micro setpoint for the next micro cycle, it first
146 updates its micro and macro cycle accounting, which determines whether a new macro cycle is to be
147 started. The current macro cycle is valid for the next micro cycle if the current spline domain is valid for
148 the time of the next micro cycle. In that case, the current spline is used to prepare the next micro setpoint.

149   If a new macro cycle is to be started, the Updater triggers the Connection thread, and the next macro
150 setpoint is retrieved from it. If a valid new macro pose is retrieved, a new spline based on this is set up,
151 and used for micro setpoint calculation. If not, due to late response or disconnect from the MA, a braking
152 spline is generated. The severity of this braking spline depends on whether the missing macro setpoint
153 was due to lateness or a broken connection. In the former case, only light braking will be generated, since
154 it is expected that the MA will resume its responsiveness. In the latter case full deceleration braking is
155 generated since the robot arm then must come to a full stop as fast as possible.

156   The Connection thread is, during operation in an MA-session, responsible for the interaction with the
157 MA. It is triggered by the Updater thread when a new macro cycle is started, which it propagates on to the
158 MA. After triggering the MA, it expects to receive a new macro setpoint some time later, before the end
159 of the newly started macro cycle. When a macro setpoint is received from the MA, a flag is set to tell that
160 a new macro setpoint is received. This flag is checked by the Updater thread, and is used to determine, if
161 the MA was within its deadline. The Updater clears the flag, preparing it for receiving the next macro
162 setpoint from the MA.

163   Timing-wise, when the macro cycle is between macro setpoint i and $i+1$, a received macro setpoint
164 is stored for use as the $i+3$'th macro setpoint; i.e. one extra macro setpoint in the stream is retained. This
165 delay is introduced to be able to correctly estimation the acceleration to target when splining toward $i+2$,
166 calculated using the central acceleration estimator over the macro setpoints $i+1$, $i+2$, and $i+3$. The
167 same principle is used for the velocity at macro setpoint $i+2$, which results in the average velocity on the
168 segments $i+1 \rightarrow i+2$ and $i+2 \rightarrow i+3$.

      The calculus of motion quantities at the macro trajectory points is based on the stream of joint positions
      received from the MA, which are assumed to arrive at regular times separated by the macro cycle time,

δt_{mac}. This stream of macro joint position vectors, $\{\mathbf{p}_i\}_i$, is the fundamental input data from the MA to the MS. Over these, the central discrete estimators of velocities and accelerations may be concisely expressed as

$$\mathbf{p}_{i+2}^{(1)} = 0.5 \left[\frac{\mathbf{p}_{i+2} - \mathbf{p}_{i+1}}{\delta t_{\text{mac}}} + \frac{\mathbf{p}_{i+3} - \mathbf{p}_{i+2}}{\delta t_{\text{mac}}} \right] \tag{1}$$

$$\mathbf{p}_{i+2}^{(2)} = \frac{\mathbf{p}_{i+3} - 2\mathbf{p}_{i+2} + \mathbf{p}_{i+1}}{\delta t_{\text{mac}}^2} \tag{2}$$

These estimated first and second derivatives of the macro joint trajectory are used for calculating the splines, which are then sampled to generate the micro trajectory positions.

Hermite Splines

Hermite spline and Hermite interpolation are named in honour of the 19th century French mathematician Charles Hermite. General treatment on multi-node Hermite splines of arbitrary order may be found in several publications and textbooks. E.g. Spitzbart (1960) focused on a general formulation for arbitrary order, while Krogh (1970) focused on efficient computation of interpolation and numerical differentiation with continuous derivative. However, course notes by Finn (2004) introduces an elegant formulation for the basis polynomials for cubic and quintic Hermite interpolation. Both have been implemented in the MS. The formalism can be concisely written, with the same enumeration of the basis functions, as

$$\tilde{p}_n(u) = \sum_{i=0}^{(n-1)/2} p_0^{(i)} H_i^n(u) + p_1^{(i)} H_{n-i}^n(u) \quad \text{for } u \in [0;1] \tag{3}$$

In Eq. (3) we assume an underlying trajectory $p(u)$ of which we know $p_\tau^{(i)} = p^{(i)}(\tau)$ for $i \in \{0..n-1\}$ and $\tau \in \{0, 1\}$, where $p^{(i)}$ is the i'th derivative of p. The basis functions for Hermite interpolation to order n are denoted $\{H_i^n\}_{i \in \{0..n\}}$. For cubic ($n = 3$) and quintic ($n = 5$) Hermite interpolation the basis functions can be found listed in Finn (2004). For completenes the coefficients are listed in matrix notation in Eqs. (4) and (5).

$$\mathbf{H}^3 = \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 3 & -2 \end{bmatrix} \tag{4}$$

$$\mathbf{H}^5 = \begin{bmatrix} 1 & 0 & 0 & -10 & 15 & -6 \\ 0 & 1 & 0 & -6 & 8 & -3 \\ 0 & 0 & \frac{1}{2} & -\frac{3}{2} & \frac{3}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & -1 & \frac{1}{2} \\ 0 & 0 & 0 & -4 & 7 & -3 \\ 0 & 0 & 0 & 10 & -15 & 6 \end{bmatrix} \tag{5}$$

In an online or real-time system a change of variable is necessary for Eq. (3) to be applicable. When the system at run-time establishes a new interpolation segment it takes the form $[t_0; t_1]$ with $\delta t_{\text{mac}} = t_1 - t_0$. Introducing the substitution on the interval $u(t) = \frac{t - t_0}{\delta t_{\text{mac}}}$, and letting $p_\tau^{(i)} = p^{(i)}(\tau)$ for $i \in \{0..n-1\}$ and $\tau \in \{t_0, t_1\}$, the directly applicable version of Eq. (3) is

$$\tilde{p}_n(t) = \sum_{i=0}^{(n-1)/2} (\delta t_{\text{mac}})^i \left(p_{t_0}^{(i)} H_i^n(u(t)) + p_{t_1}^{(i)} H_{n-i}^n(u(t)) \right) \quad \text{for } t \in [t_0; t_1] \tag{6}$$

Equation (6) has been implemented in the MS code base for both cubic and quintic Hermite splines, and can be used based on joint velocities and accelerations from the macroscopic trajectory which is fed from the MA.

It is noteworthy that a piece-wise cubic Hermite spline is a C^1-smooth function and a piece-wise quintic Hermite spline is a C^2-smooth function. With respect to generating motion trajectories, the great difference between these is the jerk; the third derivative of the position trajectory. Many motion controllers require being fed a limited-jerk trajectory, which is fulfilled by a C^2-trajectory, whereas a C^1-trajectory exhibits infinite jerk.

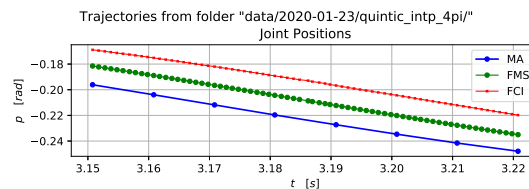



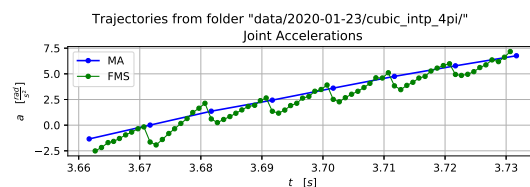
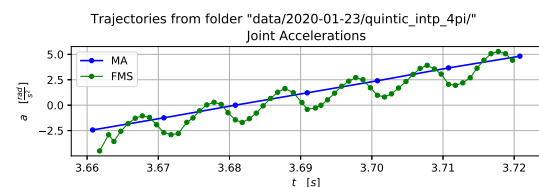
Figure 2. A selected time range of the positions in a cosine trajectory. [Image credit: Morten Lind, 2020]

(a) Acceleration for cubic interpolation. [Image credit: Morten Lind, 2020]

(b) Acceleration for quintic interpolation. [Image credit: Morten Lind, 2020]

Figure 3. Comparison of accelerations in the generated trajectories from a cosine motion using cubic and quintic Hermite interpolation.

EXPERIMENTS AND RESULTS

During development a cosine position trajectory generator have been used extensively. This section presents some results from such cosine trajectories. A cosine trajectory is used for testing, since it starts at zero speed, is cyclic, and is infinitely smooth (C^∞). In addition to the testing by a cosine position trajectory, preliminary experiments have been performed using the force-torque estimation from the FC with a Python-based framework, aimed at integrating force-feedback in a knife-cutting application. The cutting experiment does not utilize the real-time possibility of the Motion Service, but only serves to demonstrate its soundness from a simple application perspective.

Cosine Joint Motion

A simple cosine motion is generated and the motion of a single joint is observed. This will first and foremost give insight into the tracking delay in the system. The presented results defines the tracking delay from the MC perspective as the time passed from receiving a macro setpoint from the MA until a status packet from the FC shows that the joint position have been achieved.

Figure 2 show a selected time range for the position trajectory of the moving joint. The time range have been selected to easily inspect the delays from the MA over the "Franka MS" (FMS), and to the report back from the FC. The observed delay between the MA and the FMS is approximately 20 ms with the delay from the FMS to the reported FC position is another approximate 20 ms. In total, for the trajectory execution we can estimate a 40 ms delay.

It is evident that there is an inherent tracking delay of approximately 20 ms in the FC. This is affected by various filters and compliance parameters which are set at the initialization of the control loop through the FCI. Thus a lower inherent tracking delay may possible with more tight control parameters. Particularly the experiments presented here have been using a value of 10.0 Hz for the parameter `cutoff_frequency` in libfranka. This parameter controls, according to the documentation of libfranka, a first order low-pass filter. The reason for choosing a low cutoff frequency is observable as a "blackout" of two to three micro cycles near $t = 3.185$ s in Fig. 2. Such blackouts occurs generally every second, but not regularly. The reason for these blackouts have not been clarified, but they probably arise from running the system on a full desktop PC; i.e. there have been no stripping of services or other running processes on the PC to increase the real-time responsiveness.

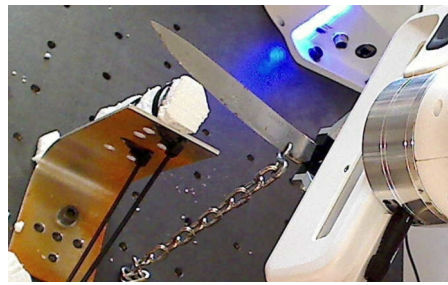
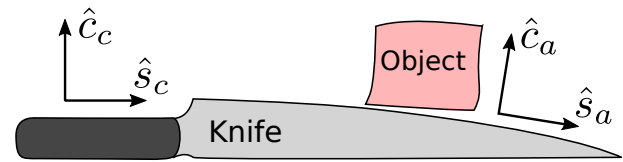
Execution of a simple cosine motion in one joint also allows the observation of the difference between using cubic and quintic Hermite splines. To this end, classes have been implemented in the FMS code base for both of these. Figure 3 shows accelerations obtained by discrete derivation of the MA and FMS

(a) Photo of the setup of robot, knife, and object. [Photo credit: Morten Lind, 2020]

(b) Illustration of controlled and actual cutting and shearing directions. [Image credit: Morten Lind, 2020]

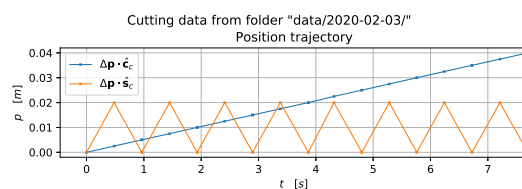
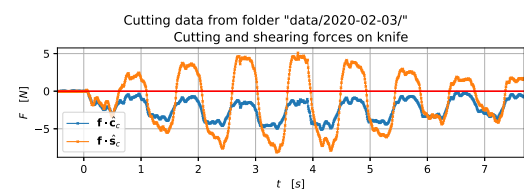
Figure 4. The setup of the knife and object. The cutting and shearing directions used for commanding the motion is aligned with the knife, whereas the actual cutting and shearing directions depend on the angle of the knife blade in the region of interaction.

(a) Commanded trajectory of the knife along the $\hat{c}_c$ and $\hat{s}_c$ directions. [Image credit: Morten Lind, 2020]

(b) Forces acting on the knife along the $\hat{c}_c$ and $\hat{s}_c$ directions.[Image credit: Morten Lind, 2020]

Figure 5. Position and force recording sampled through a cutting process. [Image credit: Morten Lind, 2020]

216   trajectories for the observed joint. Figure 3a shows the acceleration in a trajectory in a session where the
217   FMS runs with cubic Hermite interpolation and Fig. 3b shows the same type of cosine trajectory from a
218   session using quintic Hermite interpolation.

219   The cubic Hermite interpolation acceleration curve in Fig. 3a is clearly discontinuous, which is
220   consistent with its C^1-property. Correspondingly the quintic Hermite interpolation acceleration curve in
221   Fig. 3b is clearly continuous, which is consistent with its C^2-property.

Simple Cutting

223   A simple application for a cutting experiment has been set up. This is an application where the robot tool
224   is a knife and the task is to cut through a presented object. By simple cutting is meant moving the tool
225   according to a pre-calculated trajectory; in contrast to sensor-based adaptive cutting where sensor input is
226   used in the motion generation loop.

227   The application in this experiment does not exploit the ability to make real-time generation of, or
228   corrections to, the commanded trajectory. The purpose of the experiment is to do a simple demonstration
229   of the robustness of the MS implementation and to get an indication of the reliability and stability of the
230   wrist wrench estimation obtained from libfranka.

231   The minimalistic setup used in the experiment is illustrated in Fig. 4. Figure 4a shows a photo of the
232   knife held in the robot gripper and a test object; in this case a stick of EPS strapped to a bracket.

233   Figure 4b illustrates the geometric setup of the knife. The knife shaft is held clamped in the gripper,
234   and thus defines the commanded knife directions for cutting $\hat{c}_c$, which is perpendicular to the edge, and
235   shearing $\hat{s}_c$, which is parallel to the edge. The actual cutting and shearing directions are defined by the
236   knife edge at the point of interaction. These are illustrated as $\hat{c}_a$ and $\hat{s}_a$, respectively. Interpretation of the
237   data for the recorded forces in a cutting experiment is deeply dependent of the recognition of the relation
238   between these commanded and actual directions.

239   A cutting experiment is executed by moving the knife in a cyclic series of strokes, forward strokes in
240   the positive $\hat{s}_c$ direction and backward strokes in the negative $\hat{s}_c$ direction, while progressing steadily in
241   the positive $\hat{c}_c$ direction. The motion of the knife happens thus in a plane spanned by the $\hat{c}_c$ and $\hat{s}_c$. The

commanded position trajectory is shown in Fig. 5a. The origin of the positions is where the knife is held at the start of the cutting process.

The corresponding forces on the knife along the $\hat{c}_c$ and $\hat{s}_c$ directions, based on the estimated wrench obtained from libfranka in the FMS, is seen in Fig. 5b. In an ideal experiment one would expect a constant, negative cutting force in the $\hat{c}_c$ direction and a square wave shearing force, symmetric around zero, in the $\hat{s}_c$ direction. Both of these would be smoothly modulated in size by the initially increasing, and later decreasing, interaction of the knife with the object as it passes through it.

There are a couple of deviations from this ideal, qualitative behaviour. Most noteworthy is the non-smoothness of the direct cutting force. Next is the asymmetry around zero of the shear force. Both of these are easily understood by the illustration of the actual cutting and shearing directions observed in Fig. 4b.

DISCUSSION

The current major challenge regards the blackouts in the communication between MA, FMS, and FCI. For a reliable and stable system, this must be addressed by real-time hardening of the computing environment on which the application runs. However, this is fairly unrelated to the soundness of the presented approach and reference implementation.

Particularly for knife-cutting applications, the discussion about a negative bias on the cutting and shearing forces, observed in Fig. 5b, due to the deviation from the commanded directions, is interesting for future development. It shows that an edge-object-interaction observer should be developed, which will certainly be important for correct execution of the cutting process. The presented implementation will be used for future experiments in two directions involving real-time force feedback to the trajectory generation. Firstly in the direction of explicitly modelling the cutting process, based on such work as that of Reyssat et al. (2012). Secondly, on the more implicit learning approach to cutting presented by Lenz et al. (2015).

Another application area, for which the presented implementation is intended, is that of grasping of unknown or variable objects. The vision-based servo control involved with "closed-loop grasping" is executing at a fairly low frequency; in the order of 5 Hz according to Morrison et al. (2018). Thus, whereas it is expected that the required frequency and tracking delay will be a challenge for a Python-based application when it comes to real-time control of cutting, this is not expected to be a challenge for real-time grasping applications.

It was originally under consideration to use the Reflexxes[4] library developed by Torsten Kröger for the underlying interpolation mathematics in the presented Motion Service process. However, the Reflexxes version under a free software license, LGPL V3.0, only provides C^1-smooth trajectories. For generating C^2-smooth trajectories, a commercial license of Reflexxes must be purchased. The presented software is intended to be free (GPL), and thus the Reflexxes library was not chosen.

CONCLUSION

In summary, this work has

- established a re-connectable real-time motion service via the Franka Control Interface for the Panda robot;

- characterized the real-time trajectory execution performance by illustrating the tracking delay;

- demonstrated that real-time motion application from Python is possible;

- indicated the feasibility of using the external force-torque estimator provided by libfranka at the real-time application level.

ACKNOWLEDGEMENTS

Council[5] in the project "Innovative and Flexible Food Processing Technology in Norway" with project number 255596[6].

---

[4] http://reflexxes.ws
[5] https://www.forskningsradet.no/en/
[6] https://prosjektbanken.forskningsradet.no/#/project/NFR/255596

288 Thanks to Ekrem Misimi at SINTEF Ocean for general guidance and for supporting the work.

REFERENCES

290 Andersen, T. T. (2015). Optimizing the Universal Robots ROS driver. Technical report, Technical
291 University of Denmark, Department of Electrical Engineering.

292 Andersen, T. T., Amor, H. B., Andersen, N. A., and Ravn, O. (2015). Measuring and modelling delays
293 in robot manipulators for temporally precise control using machine learning. In *2015 IEEE 14th*
294 *International Conference on Machine Learning and Applications (ICMLA)*, pages 168–175.

295 Eggen, R. and Eggen, M. (2019). Thread and process efficiency in python. In *Proceedings of the*
296 *International Conference on Parallel and Distributed Processing Techniques and Applications*, pages
297 32–36.

298 Finn, D. L. (2004). Quintic Hermite Interpolation. Course notes.

299 Krogh, F. T. (1970). Efficient algorithms for polynomial interpolation and numerical differentiation.
300 *Mathematics of Computation*, 24(109):185–190.

301 Kröger, T. and Wahl, F. M. (2010). Online Trajectory Generation: Basic Concepts for Instantaneous
302 Reactions to Unforeseen Events. *IEEE Transactions on Robotics*, 26(1):94–111.

303 Lenz, I., Knepper, R. A., and Saxena, A. (2015). DeepMPC: Learning deep latent features for model
304 predictive control. In *Robotics: Science and Systems*. Rome, Italy.

305 Lind, M., Schrimpf, J., and Ulleberg, T. (2010). Open Real-Time Robot Controller Framework. In
306 Lien, T. K., editor, *CIRP Conference on Assembly Technologies and Systems*, pages 13–18, NO-7005,
307 Trondheim, Norway. Tapir Academic Press.

308 Lind, M., Tingelstad, L., and Schrimpf, J. (2012). Real-Time Robot Trajectory Generation with Python.
309 In *Workshop on Robot Motion Planning:Online, Reactive, and in Real-time*. IEEE/RJS.

310 Mihelj, M., Šlajpah, S., Čepon, P., and Munih, M. (2012). yControl - Open architecture controller for
311 Yaskawa Motoman MH5 robot. In *2012 IEEE International Conference on Control Applications*, pages
312 1051–1056.

313 Morrison, D., Leitner, J., and Corke, P. (2018). Closing the Loop for Robotic Grasping: A Real-time,
314 Generative Grasp Synthesis Approach. In Kress-Gazit, H., Srinivasa, S. S., Howard, T., and Atanasov,
315 N., editors, *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania,*
316 *USA, June 26-30, 2018*.

317 Reyssat, E., Tallinen, T., Merrer, M. L., and Mahadevan, L. (2012). Slicing Softly with Shear. *Physical*
318 *Review Letters*, 109(24):244301.

319 Schrimpf, J. (2013). *Sensor-based Real-timeControl of Industrial Robots*. PhD thesis, Norwegian
320 University of Science and Technology.

321 Schrimpf, J., Lind, M., and Mathisen, G. (2013). Real-time analysis of a multi-robot sewing cell. In *2013*
322 *IEEE International Conference on Industrial Technology (ICIT)*, pages 163–168.

323 Spitzbart, A. (1960). A generalization of Hermite's interpolation formula. *The American Mathematical*
324 *Monthly*, 67(1):42–46.