The Role of Human Motion Anticipation in a Functional Reach Task

eingereichte
Ingeneurpraxis
von

Ziyu Wang

geb. am 21.06.1992
wohnhaft in:
Martin-Behaim-Str. 5
81373 München
Tel.: 0160 91484492

Lehrstuhl für
STEUERUNGS- und REGELUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss

Betreuer: M. Sc. Matteo Saveriano, Prof. Dongheui Lee
Beginn: 29.06.2016
Abgabe: 19.09.2016

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
UNIV.-PROF. DONGHEUI LEE

20 June 2016

I N G E N I E U R P R A X I S
for
Ziyu Wang
Student ID 03637901, Degree EI

The Role of Human Motion Anticipation in a Functional Reach Task

Problem description:

With drastic growth of the computing power in recent time, robots can be utilized in many diverse ways, e.g. as a caregiver for patient with impaired balance control. A robotic caregiver is required to physically interact with the patient in a safe manner, which is usually achieved via Force or Impedance control [1]. Another fundamental requirement for a robotic caregiver is to predict the patient behaviors in order to prevent dangerous situations, e.g. falls.

In this Ingenieurpraxis work the student has to test the effectiveness of human motion anticipation in a functional reach task, where the robot correct human hand oscillation providing a light touch. The local Gaussian process approach [2] will be used to predict the human hand state during the functional reach. Predicted states will serve as references for a force controller, which guarantees a soft interaction between the patient and the robotic caregiver.

Work schedule:

- On-line data acquisition from a Xsense motion capturing suit
- Hand state prediction using local Gaussian processes
- Experimental comparison with state-of-the-art approaches (Kalman filter)

[1] C. Ott. Cartesian Impedance Control of Redundant and Flexible-Joint Robots, in *Springer Tracts in Advanced Robotics (STAR)*, 2008.
[2] D. Nguyen-Tuong, M. Seeger and J. Peters. Local Gaussian process regression for real-time model-based control, in *International Conference on Robots and Systems*, 2008.

Supervisor: M. Sc. Matteo Saveriano
Start: 29.06.2016
Delivery: 19.09.2016

(D. Lee)
Univ.-Professor

Abstract

With drastic growth of the computing power in recent time, robots can be utilized in many diverse ways, e.g. as a caregiver for patient with impaired balance control. Physical contact with the patient requires an accurate management of interaction forces, which is realized via Force Controller. The high frequency required by the Force Controller cannot be provided by the current tracking systems. The approach Local Gaussian Process aims to predict the human motion given current state to fix this issue. The model combines several local GP clusters with a weight function to achieve reasonable accuracy while staying under a certain user-defined complexity roof. Kalman Filter is, in contrast to LGP, light on memory requirement and computation, since it operates truly in real-time and does not require training data. This IP aims to test these two models, evaluate their viability and choose the suited one for our task.

Contents

Chapter 1

Introduction

1.1 Background

A strategy "light touch" for rehabilitation of patients with impaired balance has been developed, which intends to facilitate a robot caregiver as an assistant to support the patient's balance control by resting a hand on the patient without taking the patient's weight. It aims to give the patient feedback signals that allow him /her to gain enhanced body balance control. The force provided by the caregiver is small comparing to the force that would be necessary for the actual lifting of patient's body, thus, the name "light touch". The strategy has been proven to be efficient in various studies, e.g. [Jek06]. A Force Controller is regarded to be the approach in order to provide the soft and comfortable interaction needed for the strategy.

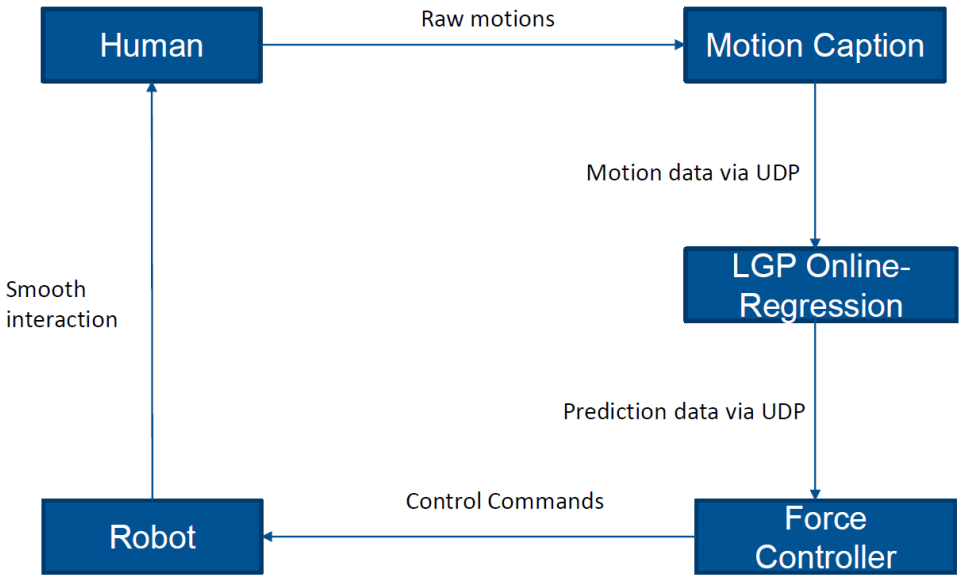
1.2 Structure Outline

An experiment in order to evaluate this strategy can be illustrated as show in Figure 1.1

The motion caption of a subject is realized with `Xsens MVN Studio`, real-time full body motion capture program. Up to 23 segments of the the body can be captured, including motion of body parts such as neck, head, shoulders and arms etc. For our evaluation, we focus on the right hand in the beginning. The details will be described in the Section 2.1.

In order to use the motion anticipation code we wrote, the raw data captured via this software has to be exported and due to the raw nature of the data, also be decoded into normal float type data, and then edited into file that is compatible with our motion prediction code. The raw data are streamed into local receiving client via UDP connection. Details are explained in Section 2.2.

Figure 1.1: Process Pipeline



The motion prediction code we use for this Ingeneurpraxis is based on my Bachelor Thesis *"Human Movements Predicion using on-line Gaussian Processes"*, which was done prior to this Ingeneurpraxis. For prediction, we use *Local Gaussian Process*, which is a modified version of the machine learning algorithm, *Gaussian Process*. This, as well as another state-of-the-art model is further elaborated on in Section 2.3.

And finally, the result submission from the local PC to the Force Controller is shortly mentioned in Section 2.4.

In Chapter 3, the two models are benchmarked against each other and a conclusion is drawn.

1.3 Related Works

For data regression, several models come to mind immediately: *Gaussian Process*, *Support Vector Machine*, *Kalman Filter*, *Locally weighted Projection Regression* etc.

Gaussian Process(GP) is first introduced in by C. E. Rasmussen and C. K. I. Williams' pioneer work [RW06]. In very simple words: a Gaussian Process can be seen as generalization of the Gaussian distribution. Instead of random variables, GP governs over functions. This model is non-parametric and flexible, scales how-

ever cubically in data size due to its kernel inversion.

Locally Weighted Projection Regression(LWPR) [VS]. It is a model suggested by S. Vijayakumar, S. Schaal, and A. D'Souza in 2005. In its core, it is a non-parametric regression model with locally linear clusters. Due to this linear nature, the model has a fantastic scaling of *O(n)* in data size and has been shown to perform well in domains of high-dimensions. The fast learning speed combined with other properties such as low requirement for training data memorization, weighting kernels adjustment based on local information and the ability do to deal with potentially redundant information make this model the standard robot control method when it comes to real-time learning.

The *Local Gaussian Process* model [NTSP09] that we use combines these two approaches and is able to deliver reasonable results while staying computationally tractable for real-time robot control tasks.

Kalman Filter [WB06] is, comparing to the previous models, a straight-forward model that works under the assumption that the system is linear. Uncertainties such as process noise as well as the measurement noise can be taken into consideration. Unlike previous models, KF does not require any training data to perform estimate and therefore, is very light on the time-consumption and a candidate for our task, since the motion we are dealing with is rather simple.

Chapter 2

Human Motion Anticipation in a functional Reach Task

2.1 Motion Caption

In order to capture a subject's motion, a tracking suit is used, which is able to track the movements of 23 joints/segments of the body. Here we try to collect the data of a simple motion as shown in Figure 2.1
Since it is a simple gesture of lifting the right hand, we mainly focused on "right hand" segment specifically while performing regression and ignore other segments such as neck, head, pelvis etc. Tracking frequency of this software is 120`Hz` and the motion has around 250-300 frames. These motions are saved in .mvn-format, which is specifically designed to be opened by MVN Studio. In order to read these data and eventually predict motions, we need to transfer these data into a local client. This will be covered in Section 2.2.

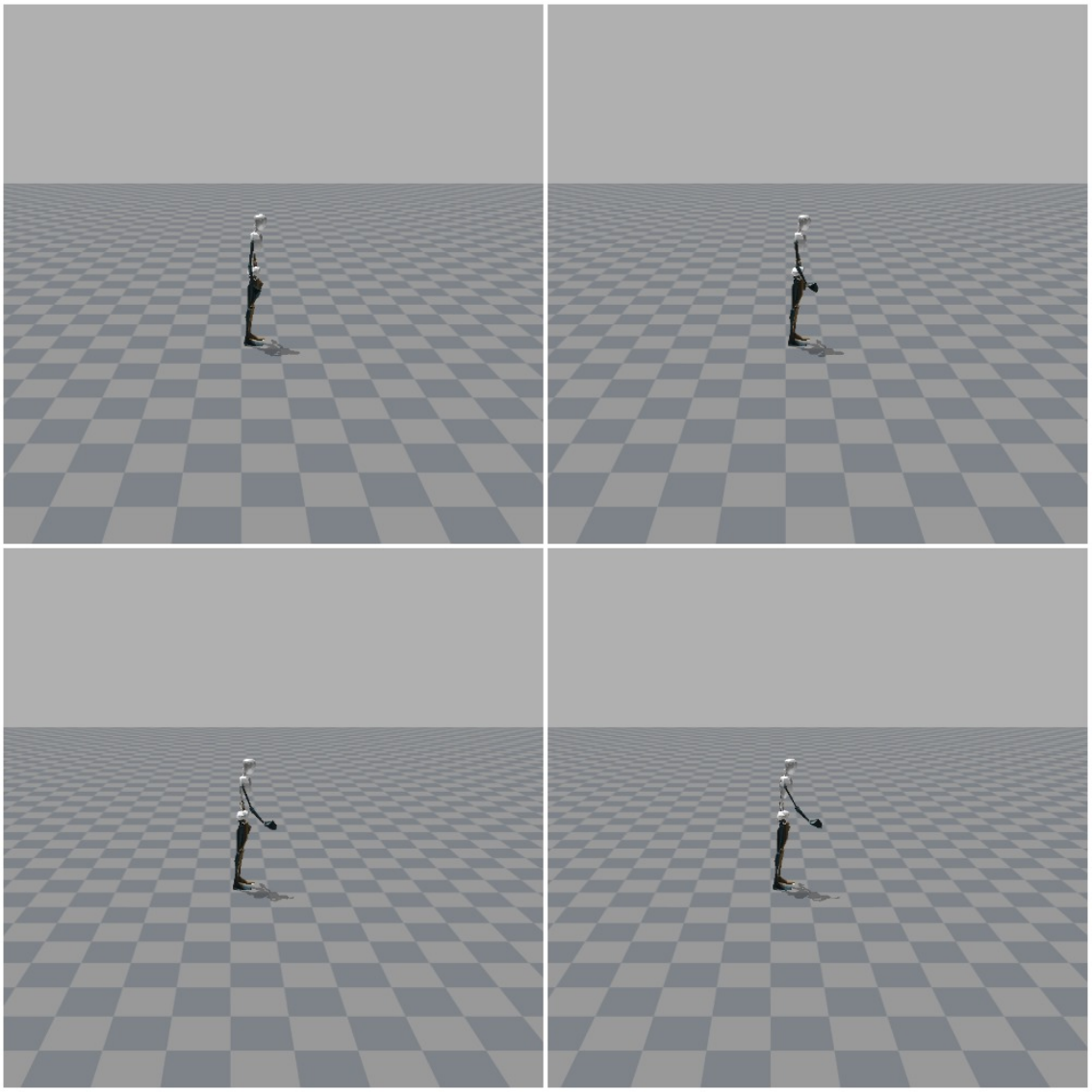
2.2 Data Processing

In this section, the data transmission between the program and a local client is described. Furthermore, the structure of the datagram is analyzed and decoded into readable data for our PC. Also, preparatory steps for motion predictions are performed. The specs of the program and the network protocol can be read in [Xse15].

2.2.1 Data Transfer

To transfer the data into local client, we use UDP connection. Unlike some other protocols, UDP connection is unidirectional and therefore does not require the receiver to answer incoming packets. This allows greater speed.
The motion tracking program itself is able to stream the data on a local host, so

Figure 2.1: Motion Demonstration



what we need is a socket that receives the data. The code `PracticalSocket` [DC02] will serve this purpose. It is a wrapper class for UDP and TCP sockets that works both on Unix and Windows platforms. It provides a simple interface and emphasizes on clarity over efficiency. Instances of the class `UDPSocket` can be created with a server-adress and -port. With the function `recv(echoBuffer, ECHOMAX)` is used for data reception. The string `echoBuffer` will be filled with the raw data received from UDP connection and the integer `ECHOMAX` defines the maximum length of the reception. The function also returns the length of the received data as integer.
One thing that caught our eyes is that sometimes, a same frame is sent repeatedly.

Should this occur frequently, it might lead to instability in the motion prediction step, since the differentiated velocity will be zero and damaging the quality of the training output overall. To avoid this, the time code of each frame is also recorded and the data will only be stored, when the frame has a different time code than the previous one.

2.2.2 Data Reception

A custom class `bodyPart` is created for storing information regarding one specific segment of the body. The informations include: segment ID that indicates which bodypart the information is from, the 3D-position in centimeter and 3D Euler-rotation in degrees.

The struct `bodyMotionframe` is further created to capture the information of all 23 segments of one time instance in form of a vector of `bodyPart`. The time code of the frame is also stored in millisecond.

In the `main` function, another vector is created, that consists of all the frames in one motion. Now that all is set up, the last step is to decode the raw data and store them into the buffers mentioned above.

2.2.3 Data Editing

The data is streamed in raw-format by the host. One single datagram-string containing all the information regarding one specific frame of the motion is transferred. Since these information are coded in different data types, we need to figure out the position of the specific information we need in the datagram-string and the data type they are encoded in. Note that the data is encoded in big-endian style and our machine works with little-endian, the proper step of reverting the bytes is necessary.

The datagram for each frame starts with a 24-byte header, which contains various information about the specific frame such as time code, sample counter, number of items etc. We only store the time code information.

After the header, the positional and angular information of the 23 segments are coded as floats. One can choose between different types of pose data to send in the MVN Studio, we use the Euler type here, which has a y-up, right-handed coordination system. Each segment contains 4 bytes segments ID in integer, 12 bytes of positional data and 12 bytes of angular data in floats, which make altogether 28 bytes per segment. Knowing this, we extract the information of each segment, decode them and store them into one `bodyPart` instance. All 23 of the `bodyPart`

instances are then pushed into a `bodyMotionframe` instance. Lastly, all of the 254 `bodyMotionframe` instances are then pushed into a vector which we will use to write the text file that is compatible with our motion prediction code.

The 3D pose data are used as training input and the corresponding velocity in three Cartesian directions are used as training outputs for three LGPs.

2.3 Motion Anticipation

The motion prediction model can be read in detail in my Bachelor Thesis, "Human Movements Prediction usding on-line Gaussian Processes" [Wan16]. Following paragraphs can be also read in the chapter "Approaches for on-line Robot Control" in the thesis.

Gaussian Process, which is explained in detail in Rasmussen's work [RW06], is used as the base model as a mean for regression for our purpose.

2.3.1 Local Gaussian Process

Gaussian Process

A GP can be used as means for regression as following:
Assuming y being the observation made at the coordinate $\mathbf{x}$ and the process is observed n times. Also assuming the prior mean is set to zero-constant as well as the covariance matrix is $\mathbf{K}\left(\Theta, \mathbf{X}, \mathbf{X}^{'}\right)$, Θ being the vector of hyper-parameters of the chosen kernel, then we can denote the GP as following:

$$f(\mathbf{X}) \sim \mathcal{GP}\left(0, \mathbf{K}\left(\Theta, \mathbf{X}, \mathbf{X}'\right)\right) \tag{2.1}$$

And the log marginal likelihood is:

$$\log p(\mathbf{f}|\mathbf{X}, \Theta) = -\frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f} - \frac{1}{2}\log\det(\mathbf{K}) - \frac{n}{2}\log(2\pi) \tag{2.2}$$

For simplicity and readability, a few terms are shortened as following: $\mathbf{K} = \mathbf{K}(\Theta, \mathbf{X}, \mathbf{X})$, $\mathbf{K}_* = \mathbf{K}(\mathbf{X}, \mathbf{X}_*)$ and $\mathbf{k}_* = k(\mathbf{x}_*)$.
Maximizing the marginal likelihood function with respect to Θ will deliver us the full specification of the GP. The first two parts on the right hand side of the equation can each be thought as the penalty terms for the fitting accuracy and for the model's complexity, respectively. With Θ given, making predictions about testing values $\mathbf{x}^*$ should work as following, given training data-set $\mathbf{X}$, the predictive distribution $p(y^*|\mathbf{x}^*, f(\mathbf{x}), \mathbf{X}) = \mathcal{N}(y^*|\mathbf{m}, \mathbf{var})$, with $\mathbf{m}$ being the posterior mean estimate and $\mathbf{var}$ being the posterior variance estimate, the predictive results m and var for a single test point $\mathbf{x}^*$ can be calculated as following:

$$\mathrm{m}(\mathbf{x}^*) = \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{y} \tag{2.3}$$

$$\mathrm{var}(\mathbf{x}^*) = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_* \tag{2.4}$$

For GP Regression, the most time consuming operation is the inversion of the kernel matrix $\mathbf{K}^{-1}$, which can take up to $O(n^3)$ flops with Gauss-Jordan. For the time-crucial operations in strategy "light touch", calculating the matrix from scratch every time a new training pair is added will not suffice. Various mathematical optimization to boost computation of the matrix inversion have been applied in our code, however, we want to increase the computation speed from the core and we want to have certain degree of control over the speed, instead of relying entirely on the data.

Local Gaussian Process

Local Gaussian Process(LGP) [NTSP09] is a combination of GP and the model *Locally Weighted Projection Regression*(LWPR) [VS].

The regression process can be divided into three parts: data-clustering, model-learning and prediction.

For data-clustering, we simply divide our training data into K subsets. If $K = 1$, this model is no different than the standard GP regression model. Each subset can be limited to a certain certain size N_{max}. In our case, we simply divide the data-set in a chronological order, but principally, the clustering function can be any arbitrary function. The parameters needed are for the LGP are therefore: amount of subsets K, maximum size of the subset N_{max} and optionally, the clustering function. Since the poor scaling of ($O(n^3)$, n is the size of the entire trainingset), which is the major drawback of GP is, is due to the inversion of the kernel matrix, splitting the data into smaller subsets leads to inversion of a much smaller kernel matrix and thus,drastic reduction of the computing time, both for adding data and calculating regression. By limiting the subsets to a fixed size, the total complexity is $O(MN^3)$, where M is number of relevant models for the prediction and N is the size of the subsets. In most cases, $MN^3 \ll n^3$.
The model-learning complexity of each local model can be reduced from $O(n^3)$ to $O(n^2)$ with some optimization routines[See04].
A commonly used function for weight calculation is the *Radial Basis Function*:

$$w_k(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_k\|^2}{2l^2}\right) \tag{2.5}$$

l defines the *characteristic length-scale* and $\mathbf{c}_k$ the *centroid* of k-th local model.

Assuming points belonging to the same region are more informative for the prediction, this function is our function of choice to determine the relevance w_k of a certain cluster is to a query point. When adding a new point $\mathbf{x}$, the cluster i with the highest relevance w_i will be chosen. If the cluster size exceeds a certain limit, an old point from the data-set will be removed in order to keep the size of the subset. A weight threshold w_{th} is introduced, so that if none of the cluster has a sufficiently high weight, a new cluster will be created with with $\mathbf{x}$ as its centroid. These two steps are can be summarized in Algorithm 1.

For prediction, we can apply the formulas (2.3) from GPR for local regression:

$$\bar{y}_k(\mathbf{x}^*) = \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{y} \tag{2.6}$$

and for final regression:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{k=1}^{M} w_k \bar{y}_k(\mathbf{x})}{\sum_{k=1}^{M} w_k} \tag{2.7}$$

where M is number of relevant models for the prediction. This is summarized in Algorithm 2.

Algorithm 1 Partitioning the data and updating the kernel

Require: New observation data $\{\mathbf{x}_{new}, y_{new}\}$
  for $k = 1$ **to** M = amount of all clusters **do**
    Calculate the weights of these clusters
    $w_k = f(\mathbf{x}_{new}, \mathbf{c}_k, \Theta_k)$
  end for
  Choose the cluster with the highest weight
  $w_{max} = \max(w_1, ..., w_M)$
  if $w_{max} > w_{th}$ **then**
    Add the data pair to the cluster with the highest weight
    $\mathbf{X}_{new} = [\mathbf{X}, \mathbf{x}_{new}],\ \mathbf{y}_{new} = [\mathbf{y}, y_{new}]$
    Update the centroid of the cluster
    $\mathbf{c}_{new} = \text{mean}(\mathbf{X}_{new})$
    Update kernel inverse with techniques
  else
    Build new cluster with $\mathbf{x}_{new}$ as centroid
    $\mathbf{c}_{M+1} = \mathbf{x}_{new},\ \mathbf{X}_{M+1} = [\mathbf{x}_{new}],\ \mathbf{y}_{M+1} = [y_{new}]$
  end if

Algorithm 2 Prediction using LGP

Require: test point $\mathbf{x}^*$, amount of clusters M
  for $k = 1$ **to** M **do**
    Calculate the weights of the clusters
    $w_k = f(\mathbf{x}^*, \mathbf{c}_k, \Theta_k)$
    Calculate the mean $\bar{y}_k$ using parameters from the local Gaussian Process with eq.(2.3) and eq.(2.6)
    $\bar{y}_k = m_k(\mathbf{x}^*) = \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{y}$
  end for
  Calculate the weighted mean prediction $\hat{y}$ from the local clusters with eq.(2.7)
  $\hat{y}(\mathbf{x}^*) = \frac{\sum_{k=1}^{M} w_k \bar{y}_k(\mathbf{x}^*)}{\sum_{k=1}^{M} w_k}$

2.3.2 Kalman Filter

Kalman Filter [WB06] is also a linear model capable of making estimations about a dynamic system in real-time. Knowing the system dynamic and the time step beforehand, the model is able to make educated guess on a certain measurement, concerning factors such as process noise and measurements noise.
Assume we have a linear model with following dynamics with $x_k \in \mathbb{R}^n$ as the state and $z_k \in \mathbb{R}^m$ as the measurements:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.8}$$

$$z_k = Hx_k + v_k \tag{2.9}$$

In our case, we set the state and measurement as:

$$x_k = \begin{bmatrix}\text{position}\\ \text{velocity}\end{bmatrix}, \qquad z_k = \begin{bmatrix}\text{position}\end{bmatrix}$$

so that our system dynamic matrices in eq. (2.8) and eq. (2.9) look as following:

$$A = \left[\begin{array}{c|c} I & \Delta t I \\ \hline 0 & I \end{array}\right], \qquad H = \left[\begin{array}{c|c} I & 0 \end{array}\right]$$

with $x_k \in \mathbb{R}^6$, $z_k \in \mathbb{R}^3$, $A \in \mathbb{R}^{6\times6}$, $H \in \mathbb{R}^{3\times6}$ and Δt as time step in seconds. Since we only measure and try to predict in our work, we will set control input u_k to zero.

w_k and v_k represent the process and measurement noise, respectively. We assume them to be Gaussian distributed:

$$p(w) \sim \mathcal{N}(0, Q), \qquad p(v) \sim \mathcal{N}(0, R)$$

with process noise covariance Q and measurement noise covariance R.

Let $\hat{x}_k^-$ and $\hat{x}_k$ be our *a priori* and *a posteriori* estimates respectively, e_k^- and e_k the respective errors, the error covariances can be calculated as:

$$P_k^- = E[e_k^- e_k^{-T}], \qquad P_k = E[e_k e_k^T]$$

With these system dynamics, we can predict from k-th state from $k-1$-th state a priori as following:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}, \tag{2.10}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.11}$$

To minimize the a posteriori error covariance P_k, we compute the *Kalman Gain* K

$$K = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{2.12}$$

the updated P_k becomes:

$$P_k = (I - K_k H) P_k^- \tag{2.13}$$

And the new a posteriori estimate can be be calculated as:

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{2.14}$$

With equations eq.(2.10) - eq.(2.14), a closed loop is built so that this procedure can be called recursively.

2.4 Data Submission

After performing the LGP regression as explained in the previous section, one more step awaits for the motion prediction: to integrate the predicted velocity into the previously measured pose data. Since the code is supposed to work generically for every subject from every initial position, we also subtract it from all the motion data received in real-time.

Since the UDP is unidirectional, we need another socket for the regression data submission to the Force Controller. For this, another instance of the class `UDPSocket` is established and will facilitate as a sender. As mentioned in the Section 2.2, the local small-endian style has to be reverted to big-endian before sending it through network, the necessary steps are performed. The data reception on the side of the Force Controller, however, has not yet been evaluated and is among the future works.

Chapter 3

Experimental Results

3.1 Benchmark Results

After the steps in Chapter 2 have been performed for LGP estimate results, the results of one take have been evaluated. The plots are shown in Figure 3.1.

We first reconstruct the trajectory and compare it with the original measurement. See Figure 3.2a.

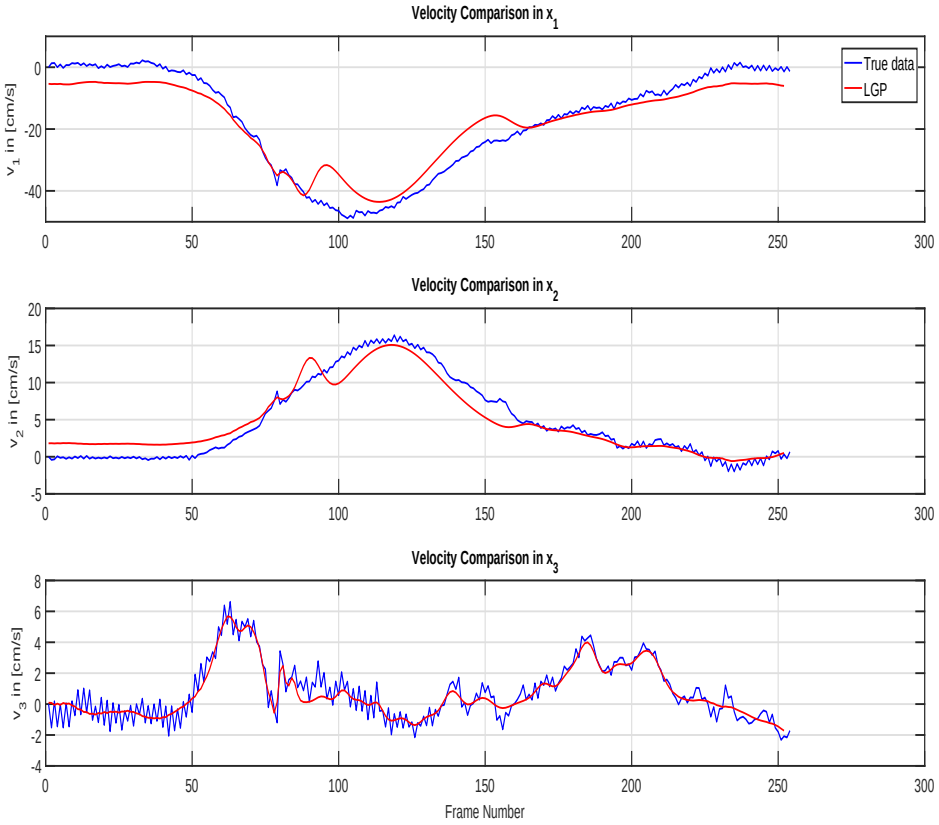
To see how it stands with state-of-the-art, we also use the Kalman Filter code developed by Greg Welch and Gary Bishop, which is an implementation of the article [WB06].
Since Kalman Filter only predicts upon the current measurement and works without memory and thus requires no specific modification for each trial, the same system dynamic matrix will be used for every plot.

Using the first take as training data for out LGP, we test it on motion data of another take, results can be seen in Figure 3.2b.

To see if it works on a different person, we trained on the motion data of one person and perform regression on a second person. The results are plotted in Figure 3.2c

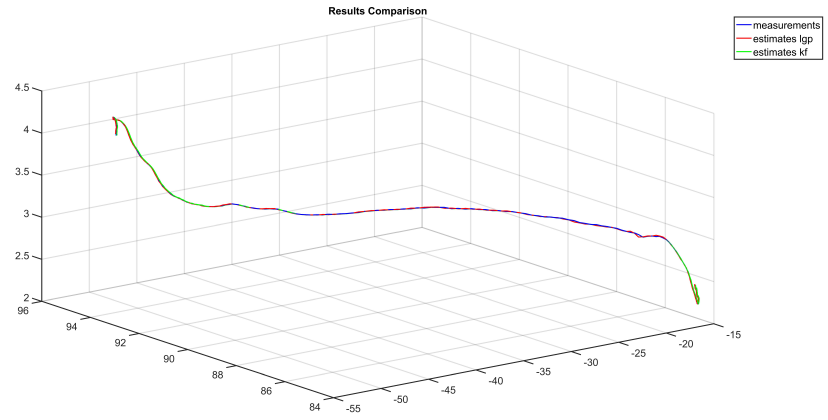
The the consumed time for each model has also been logged. Since the Kalman Filter is a local model, it requires no pre-logged training data, nor does it need the differentiated velocity to perform regression, thus it has no memory requirement on the system and is very time-saving, so that with the standard timing function from c++ library `std::clock()`, there is barely any time consumed for the update step, resulting in 0ms of computation. LGP however does need to consider the training data and perform somewhat time-consuming operations such as matrix inversion. Overall for each regression step, approx. 3ms seconds are needed (1ms for each LGP-model, which is responsible for one direction).

Figure 3.1: Velocity Comparison

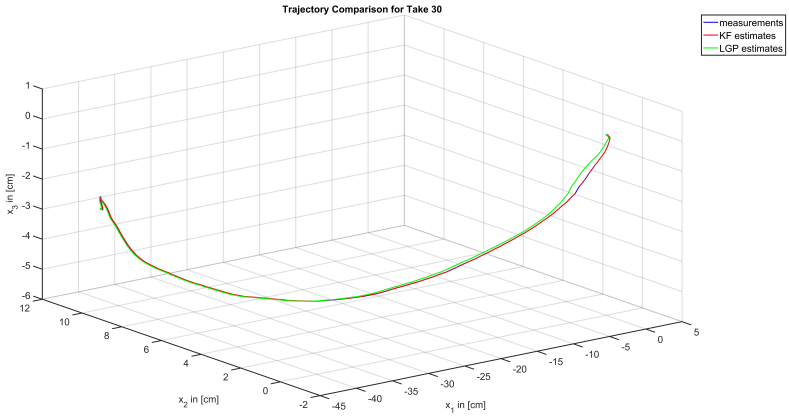


The blue line shows the true velocity, whereas the red line corresponds to the predicted velocity using LGP with 3 clusters and RBF as weighting function. x-axis is the number of frame in a chronological manner and y-axis is the velocity in the respective direction in [cm/s].

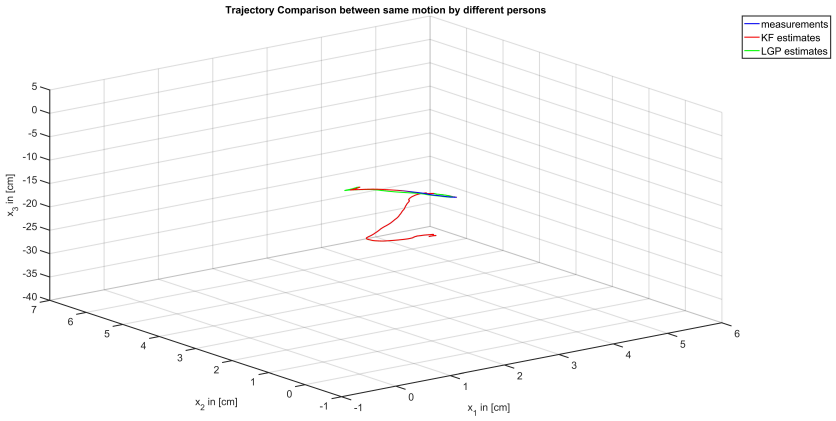
Figure 3.2: Trajectory Comparisons



(a) The LGP is trained and tested on the same take from the same person.



(b) Here, the LGP is trained on motion data of a different take of the same person



(c) For this plot, the LGP is trained with motion data from one person and tested on a second one.

In these figures, the blue line shows the original measured data, the red line shows the KF estimates and the green line the LGP estimates.

Table 3.1: RMSE Comparison

For Figure 3.2a	In x_1	In x_2	In x_3
LGP in [cm]	0.0421	0.0123	0.0062
KF in [cm]	0.0326	0.0091	0.0014
For Figure 3.2b	In x_1	In x_2	In x_3
LGP in [cm]	0.0421	0.0123	0.0062
KF in [cm]	0.0326	0.0091	0.0014
For Figure 3.2c	In x_1	In x_2	In x_3
LGP in [cm]	0.0421	0.0123	0.0062
KF in [cm]	0.0326	0.0091	0.0014

One can easily observe from the plots that the estimated data from both LGP and KF are very close to the original line so that the visual difference is barely noticeable. Therefor the exact Root-Mean-Square-Error of both models are logged and compared to each other.

3.2 Conclusion

For simple gestures and time-crucial operations, there is no argument against using the Kalman Filter over LGP due to its fast computation, easy-to-use interface and its lightness for since it requires no prior training data, therefor also no memory requirement for the system. Since the Force Controller has to operate around 1ms, Kalman Filter will be the go-to model for our task.

Despite more time-consumption and memory requirement and a somewhat complex setup procedure, the strength in LGP should lie in its adaptability, flexibility and genericity. The more training data it possesses, the more accurate the estimation would be, this has been proven in other studies, but in our task, the emphasis lies in time-consumption and Kalman Filter will be the better choice.

Chapter 4

Summary and Future Works

To sum up everything achieved in this Ingeneurpraxis so far:

- motion of a simple gesture (lifting right hand) has been captured repeatedly through multiple takes, data is stored in raw format,
- data are streamed via UDP connection on a local host,
- a UDP client has been created for the data reception,
- necessary steps for reading specific data at the right position in a right way have been performed,
- pre-processed the data to be zero-mean
- the training data are then stored locally in an instance of a custom class,
- the data in the instance containing all the information of the motion are written in a file that is compatible with the motion prediction code,
- the online prediction of the first take is performed using KF and LGP,
- on-line the motions of a different repetition using KF and LGP and
- encoding the data into correct format and send them via UDP connection to the Force Controller

The future works might include:

- experimentally evaluating the results with Force Controller,
- evaluation of a more complex motion and
- real-world task cooperated with a KUKA robot.

List of Figures

Bibliography

[DC02] M. J. Donahoo and K. L. Calvert. *TCP/IP Sockets in C: Practical Guide for Programmers.* Elsevier Science, 2002.

[Jek06] J. Jeka. Light Touch Contact: Not Just for Surfers. *The Neuromorphic Engineer. A Publication of INE-WEB.ORG*, 2006.

[NTSP09] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model Learning with Local Gaussian Process Regression. *Advanced Robotics*, 2009.

[RW06] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2006.

[See04] M. Seeger. Low rank updates for the cholesky decomposition. Technical report, 2004.

[VS] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *in Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 1079–1086.

[Wan16] Z. Wang. Human movements prediction using on-line gaussian processes. Bachelor Thesis, Technical University of Munich, 2016.

[WB06] G. Welch and G. Bishop. An Introduction to the Kalman Filter. *University of North Carolina at Chapel Hill, Department of Computer Science*, 2006.

[Xse15] Xsens Technologies B.V. *MVN Studio real-time network streaming*, 2015. Protocol Specification.

License