Semester project

Gaussian Process Regression for State Estimation

Student:
André Gomes

Supervisors:
Prof. Dimitrios Kyritsis
Paul-Arthur Dreyfus

Abstract

This semester project investigates the use of Gaussian Process Regression methods to estimate the quality of a part after milling operations.

The milling process is studied to understand the key parameters at play. This provides knowledge on which timesteps bring relevant information to increase the prediction performances at a given time. It also confirms from a theoretical viewpoint that power and vibration measurements extracted from the simulation are meaningful for the prediction problem.

The results obtained show the importance of the different preprocessing steps implemented. Cleaning of the dataset and data-augmentation with previous timesteps lead to increase of the predictive performances of the models trained. Feature selection doesn't improve the results but significantly reduces the computation time. This indicates that for time critical predictions this step should not be neglected.

The kernels used in this project are the most common found in the literature due to limited prior knowledge. Further research on multi-task learning and the relationship between the milling parameters and the quality estimates should therefore be carried out. This would allow the use of Gaussian Processes at their full potential by constructing kernels shaped specifically for the milling operations. For real-time applications, approximate methods with uncertain inputs should be explored to overcome the limitations of exact Gaussian Processes.

Spring Semester 2020

Contents

1 Introduction **4**

2 State-of-the-art - Gaussian Processes in machining **5**

2.1 Energy prediction model for a milling machine tool . . . 5

2.2 Prediction of surface roughness in end face milling . . . 5

3 Problem definition **6**

3.1 Milling process . . . 6

3.2 Quality definition . . . 7

3.3 Simulation . . . 8

3.3.1 Purpose . . . 8

3.3.2 Main outlines . . . 8

3.3.3 Measured information at each timestep . . . 9

3.3.4 Computing the quality estimate . . . 9

3.3.5 Simulated operations . . . 9

3.4 Quality prediction . . . 10

3.4.1 Theoretical relation between quality and power . . . 10

3.4.2 Instantaneous cutting teeth . . . 11

3.5 Window size . . . 11

3.6 Relating multiple timesteps to quality estimates . . . 14

4 Preprocessing **14**

4.1 Loading . . . 14

4.2 Cleaning . . . 14

4.3 Data augmentation . . . 15

4.4 Splitting . . . 15

4.5 Features selection . . . 15

4.6 Scaling . . . 17

4.6.1 Standardization . . . 17

5 Gaussian Process Regression **18**

5.1 Prediction with Bayesian inference . . . 18

5.2 Limitations . . . 19

6 Kernels or covariance functions **19**

6.1 Lengthscales and automatic relevance determination . . . 19

6.2 Scaling factor . . . 19

6.3 Squared-exponential kernel . . . 20

6.4 Rational quadratic kernel . . . 20

6.5 Matérn kernel . . . 20

6.6 Adding and multiplying kernels . . . 20

6.6.1 Linear and periodic kernels . . . 21

7 Optimisation of the kernel parameters **22**

7.1 Loss function - Marginal log likelihood . . . 22

7.2 Optimisation solver - Stochastic gradient descent . . . 22

7.3 Score metrics . . . 23

7.3.1 Mean absolute error . . . 24

7.3.2 Mean absolute percentage error . . . 24

7.3.3 Mean squared error . . . 24

7.4 Cross-validation . . . 24

8 Implementation **25**

8.1 Python and GPytorch library . . . 25

8.2 Algorithm . . . 25

8.3 Randomness . . . 26

9 Results **26**

9.1 Kernels . . . 27

9.1.1 Matérn . . . 27

9.1.2 Rational quadratic . . . 28

9.1.3 SE . . . 28

9.1.4 Comparison between kernels . . . 28

9.2 Naive approaches . . . 29

9.3 Target shuffling . . . 29

9.4 Default preprocessing and kernel . . . 30

9.5 Feature selection . . . 30

9.6 Cleaning the zero power inputs . . . 31

9.7 Data-augmentation . . . 31

9.8 Adding *Random* dataset . . . 32

10 Future prospects **33**

10.1 Prior knowledge on the problem 33

10.2 Approximate methods 33

10.3 Uncertain inputs 34

11 Conclusion **34**

1 Introduction

Increasing the quality of manufactured parts requires better control of the tool. To correctly tune control parameters, precise quality estimates are necessary.

This project starts with an analysis of how the milling affects the quality of the part manufactured. This analysis is then used to relate how the quality estimates generated by the simulation are related with the position of the tool and all the information measured at multiple timesteps.

Then, from the generated sets of data, the objective is to compute quality estimates of the manufactured part using a Gaussian Process Regression (GPR) method. The focus of this work is not directly on obtaining the best quality estimates results for a given dataset. This would require an important amount of time spend of simulating with different parameters using a grid search approach to find a quasi-optimal set of optimisation parameters (optimiser, learning rate, ...) in addition to the myriad of model choices and their related hyperparameters. This work rather focuses on understanding what are the important steps in order to find a good model, from the standard good practices of Machine Learning to more specific aspects related to the problem at hand.

The problem definition and preprocessing of the datasets done for this project are carry out in close collaboration with Pierre-Emmanuel Terrier. The related sections are mostly the same in both reports as no distinction had to be made to later implement the algorithms studied. While this project investigates Gaussian Processes (GP) for quality estimation, Pierre-Emmanuel work focuses on the use of Mixture Density Networks approaches [1].

2 State-of-the-art - Gaussian Processes in machining

It is summarised in this section two interesting applications of GP for prediction in the machining domain. The first relates to the prediction of the energy required during milling operations and the second to the roughness prediction based on three common milling parameters.

2.1 Energy prediction model for a milling machine tool

This research on real-time energy prediction for a milling machine tool shows that prediction models can be constructed using GPR to predict energy consumption of a machining tool [2]. This is done by investigating the fit of the GPR prediction on various operations of a CNC milling machine (face milling, contouring, air cut in x-y, etc).

It gives an interesting feedback on what are the important steps to perform relevant prediction for milling machines. The first step is how the data is obtained. As the process parameters (feed rate, spindle speed, depth of cut, etc) are produced with different combinations of machining parameters, a special attention should be given to the design of the experiments. This is necessary to ensure correct combination of machining parameters for each of the process parameters.

The data obtained through the experiment is then filtered to ensure that the time series data were statistically of good quality. In this work, some operations (NC code blocks) were too short (under 3 seconds) to be relevant compared to longer ones. Further more, as the data obtained covered different operations, they are categorize for each operation. This allows to train models specifically for one operation.

Using the training dataset, validation is done to select the optimum input features for each operation. This step gives information on which features are relevant to predict the energy consumption. It is interesting to note that including all the available input features doesn't always mean achieving the lowest error ratio.

The results from this research show that GPR can accurately predict energy consumption from multiple input features using only a restricted number of experimental data. The burden of the computation and data storage to train the models are also discussed. These are well-known limitations of applying GP in regression tasks. Further investigation on implementation of a real-time adaptive learning algorithm is proposed.

2.2 Prediction of surface roughness in end face milling

The objective of this research is to construct a GP model to predict surface roughness on end face milling operations [3]. The prediction should be based on the influence of some cutting parameters. The trained model should then be used to construct 3D-maps of the surface roughness based on these parameters, allowing an easy selection process of the optimal milling parameters.

With an experimental setup, 48 experiments of end face milling are performed. Three milling parameters are investigated, depth of cut (0.2 to 0.6 [mm]), feed rate (100 to 275 [mm/min]) and spindle speed (2500 to 6000 [rpm]). 36 experiments are used to train a GP model with a Matérn covariance function. The covariance function is designed with signal variance scaling, smoothness $\nu = 1.5$, one lengthscale per input feature with an addition of noise to the outputs. The lengthscales and noise parameters are optimised through optimisation of the marginal log likelihood.

The paper emphasises on how the changes in the cutting parameters studied affect the surface roughness in a coupled, nonlinear manner. Among them the depth of cut is the parameter that has the larger impact on the roughness. According to further results, the tool vibration is an important source of surface defect and should not be neglected.

From the GPR perspective, with an accuracy prediction of about 84.3%, the trained model is used to construct 3D-maps of surface roughness with respect to the studied parameters. This could allow to select the milling parameters without the costly trial-and-error procedure.

3 Problem definition

The purpose of this section is to define the problem statement to have a common basis and try to visualize as best as possible what was expected and how this has been answered.
This project aims to predict the quality of a workpiece after a milling operation knowing the instantaneous tool parameters during the operation.

3.1 Milling process

All the work done during this project is related to a specific operation realized in manufacturing: milling. Milling is the process of machining using rotary cutters to remove material by advancing a cutter into a workpiece. This may be done varying direction on one or several axes, cutter head speed, and pressure. Milling covers a wide variety of different operations and machines, on scales from small individual parts to large, heavy-duty gang milling operations. It is one of the most commonly used processes for machining custom workpieces to precise tolerances.
Milling is a cutting process that uses a milling cutter to remove material from the surface of a work piece. As opposed to drilling, where the tool is advanced along its rotation axis, the cutter in milling is usually moved perpendicular to its axis so that cutting occurs on the circumference of the cutter. As the milling cutter enters the work piece, the cutting edges of the tool repeatedly cut into and exit from the material, shaving off chips from the workpiece with each pass. The cutting action is shear deformation.
The milling process removes material by performing many separate, small cuts. This is accomplished by using a cutter with many teeth, spinning the cutter at high speed, or advancing the material through the cutter slowly; most often it is some combination of these three approaches. The speeds and feeds used are varied to suit a combination of variables. The speed at which the piece advances through the cutter is called feed rate.
There are two major classes of milling process:

- Face milling: the milling cutting is placed perpendicular to the workpiece. The milling cutting is essentially positioned *face down* towards the top of the workpiece. When engaged, the top of the milling cutting grinds away at the top of the workpiece to remove some of its material.

- Peripheral milling: the milling cutter is placed parallel to the workpiece. In other words, the milling cutter is positioned so that the sides of the cutter grind away at the top of the workpiece.

In this project a specific type of peripheral milling is considered: plain milling. For plain milling (also called slab milling) the width of the tool is higher than the width of the workpiece. It is possible to have either conventional (up milling) or climb milling (down milling) where the difference is the cutting direction of the chip. An illustration of the process is shown in Figure 1.

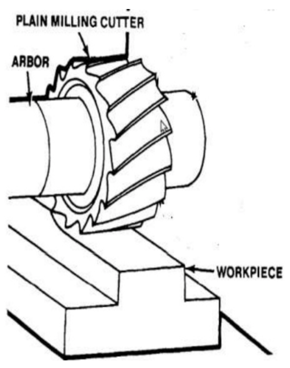


Figure 1: Scheme of peripheral milling operation (down milling).

It is important to define the cutting parameters [4] used along this project.

- Z_{eff} $[-]$: number of teeth
- f_z $[mm]$: feed per tooth
- n $[rpm]$: spindle speed
- v_c $[mm/min]$: cutting speed
- v_f $[mm/min]$: feed rate
- R $[mm]$: tool radius
- ϕ_{cut} $[rad]$: cutting angle of one tooth (within the part)
- a_e $[mm]$: radial depth of cut
- a_p $[mm]$: axial depth of cut
- k_c $[N/mm^2]$: specific cutting force
- η $[\%]$: cutting efficiency

3.2 Quality definition

The main goal is to predict the quality of the workpiece after milling. The term quality has to be defined to clearly visualized what is meant.
The 2D scheme shown on Figure 2 represents the path that the milling tool goes through at different timesteps as well as the workpiece currently machined. It has to be imagined that this happens all along the z axis considering the helical shape of the tool.
In the ideal case the surface quality at the end of the process should be as displayed by the red line. However, it can clearly be seen that according to the tool parameters the surface quality at the end of the operation can vary and the resulting part can hence have flatness

inhomogeneities.
Thus, speaking of quality prediction, the goal is here to predict the space between the red line and the black area of the circle.
It has to be noted here that the presented case is semi-ideal since it is assumed that the surface quality before milling is perfect (perfect roughness). In reality the initial workpiece will never be as represented on the scheme but in this case, a preliminary study has been done and it appears to be very complicated to predict the quality. Indeed, the depth of cut could be predicted but the quality itself would remain unknown. A reference (corresponding to the orange line) or the exact shape of the surface of the workpiece before milling is needed.

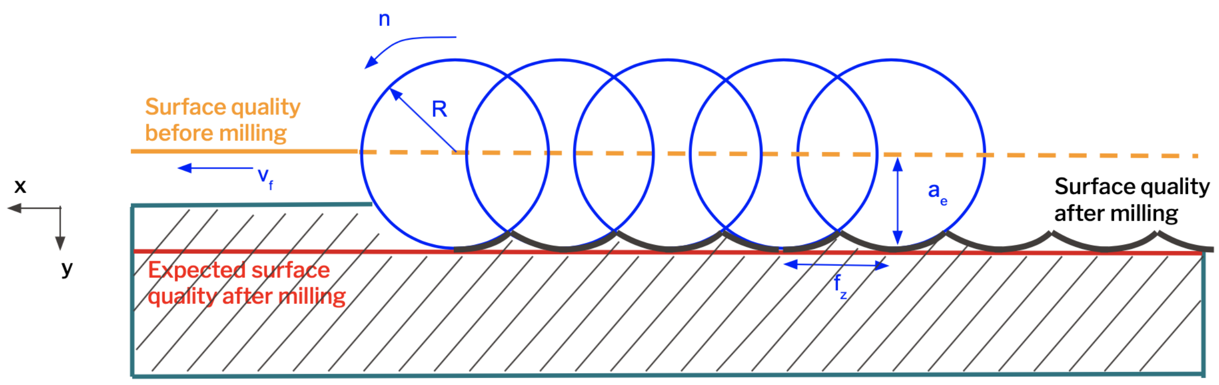

n
R
Surface quality before milling
v_f
x
y
a_e
Surface quality after milling
Expected surface quality after milling
f_z

Figure 2: 2D scheme of the quality characterization for a plain milling tool operation.

3.3 Simulation

The simulation used to extract the datasets is first described in the Master Project from Edouard Le Goas (2019) [5]. In parallel of the present project, Francisco Montoliu Cervero is further improving the simulation while working on the quality control using reinforcement learning [6]. The datasets used here are kindly provided from his work.

3.3.1 Purpose

The simulation at hand serves as a virtual twin of a milling machine. The use of such virtual machines is essential to create adequate algorithms for the control of real machines. Using the simulation, the goal is to extract information that can be measured in real-time on milling machines along with quality estimates of the surface of the workpiece being machined. In reality, obtaining quality estimates information is not easily achievable. Here, as the milling operations are simulated, this crucial information can be extracted along with machining parameters that lead to such quality. The data collected can then be used by machine learning algorithms to train estimation models. These models are then able to predict the quality estimate based only on real-time measured information from the sensors of the real machine. Finally the goal would be to have real-time quality estimate that can be used in the control loop of the system.

3.3.2 Main outlines

The simulation of the milling process models both the tool and the workpiece being milled.

The tool is defined as a non-rigid 3D beam-like body. It's dynamical behaviour is modelled by the Euler-Lagrange equation. Due to its aspect ratio, only deformations along the z-axis (the axis of the tool) are considered. It's behaviour is therefore given by:

$$EI\frac{\partial^4 w}{\partial z^4} + \mu\frac{\partial^2 w}{\partial t^2} = q(z) \qquad (1)$$

where w describes the deflection of the tool, E is the elastic modulus, I is second moment of area of its cross-section and $q(z)$ is the external load applied along the z-axis. The tool is modelled with clamped-free limit conditions, having one extremity hold by the machine and the other one free.

The tool is subjected to deformations when forces are exerted on it. The workpiece reaction forces applied on the tool are highly non-linear and induce non-negligible vibrations of the tool. The dynamics of the tool have an large impact on the final surface quality. The vibrations induced by the deformation of the tool are therefore modelled in the simulation by approximating the force exerted on each tooth of the tool.

3.3.3 Measured information at each timestep

At each timestep, the simulation records the following:

- t $[s]$: timestamp
- x, y, and z $[mm]$: position of the tool
- v_x, v_y and v_z $[m/s^{-2}]$: vibrations of the tool along the three axis
- p_x, p_y and p_z $[W]$: power to translate the tool along the three axis
- p_s $[W]$: spindle power

These information are the one that could be expected to be measured on real milling machines with the appropriate sensors.

3.3.4 Computing the quality estimate

At each timestep, the quality is computed as the mean difference between the final y-position of each discretized point being milled at that timestep (points having their position updated) and the targeted position for that same points. The final y-position is taken at the end of the simulation, as a point may have its position updated multiple times.

3.3.5 Simulated operations

Two types of operations are simulated and used to train the models.

Rotated operation: The tool moves at constant speed along the x-axis. The tracked y-coordinate of the tool remains constant and the z-axis corresponds to the tool axis. The workpiece surface being milled is not parallel to the trajectory of the tool. It's rotated around the x and z-axis (Figure 3).

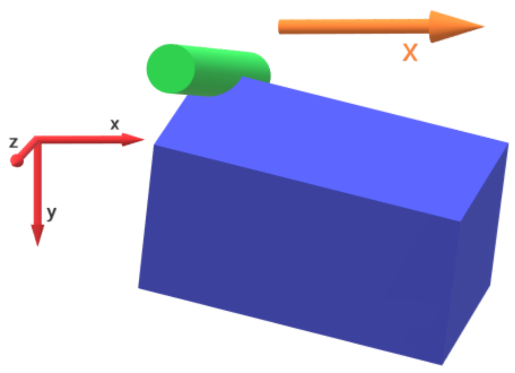


Figure 3: Illustration of the rotated milling operation. The tool, in green, moves only along the x-axis (direction of the red arrow). The part, in blue, is rotated around the x and z-axis.

Random operation: The tool moves at constant speed along the x-axis. Additionally, random y-positions (between -500 and $+500$ $[\mu m]$) are tracked at each timestep, leading to additional random movements along the y-axis. The workpiece surface being milled is in the same plane as the tool axis (Figure 4).

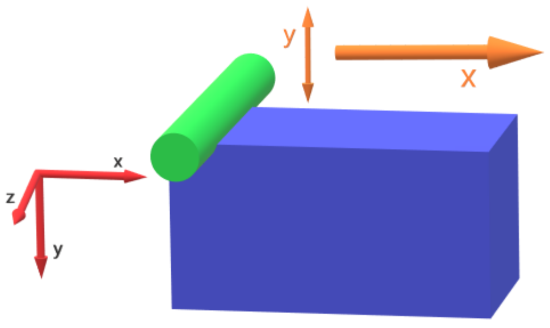


Figure 4: Illustration of the random milling operation. The tool, in green, moves at constant velocity along the x-axis with simultaneous random movement along the y-axis (direction of the red arrows).

3.4 Quality prediction

3.4.1 Theoretical relation between quality and power

One possible output to retrieve from the simulation is the instantaneous power of the milling tool. Hence, a very important part of the project is to find whether there is a direct relation between the power of the tool and the quality of the part. Indeed, the idea behind that is to use this power as an input feature for the machine learning algorithms.

Using the basics formulas of milling machines, it is possible to have the cutting instantaneous power consumed during milling as a function of the radial and axial depth of cut (see Equation 2) [4].

$$P_c = \frac{a_e \times a_p \times v_f \times k_c}{60.10^6 \times \eta}[W] \tag{2}$$

Having, such a relation allows to theoretically understand the direct relation between the cutting power and the quality because the quality is related to the depth of cut. Hence, using the power as a feature for machine learning algorithms in this case is consistent.

3.4.2 Instantaneous cutting teeth

At a given timestep t_i, it is important to define the number of teeth currently cutting the workpiece. Indeed, as it will be seen later, this is essential for the number of previous timesteps to be considered when trying to estimate the quality computed at this specific timestep. Two parameters can influence the current number of teeth cutting:

- The number of teeth Z_{eff} that the tool has. Indeed, for instance if the tool has only three teeth, there will be intervals when the tool is not cutting the workpiece and thus giving rise to a lot of timesteps where the power measured by the simulation is zero.
  The number of teeth chosen for the simulation is $Z_{eff} = 8$. It has been found to be a good trade-off between the number of teeth and the number of timesteps when the power is zero.

- The position of the tool axis with respect to the workpiece surface. A first configuration is when the tool rotation center is upper with respect to the workpiece and a second configuration when it's lower. In the first case the teeth will be less in contact with the workpiece than for the second case.
  For the simulation the tool center of rotation is positioned at the surface of the workpiece before milling.

3.5 Window size

From the scheme presented in Figure 5, it can clearly be seen that for a quality measurement computed at given timestep t_i represented by a red cross, the quality of this point will not only depend on the tool parameters at this specific timestep but also from what happened previously. Indeed, this is illustrated by the cutting paths that the tool went trough represented by grey circular arcs. Also, the dashed line displays the number of paths that were required to obtain a milled part from the rough workpiece for the considered point.

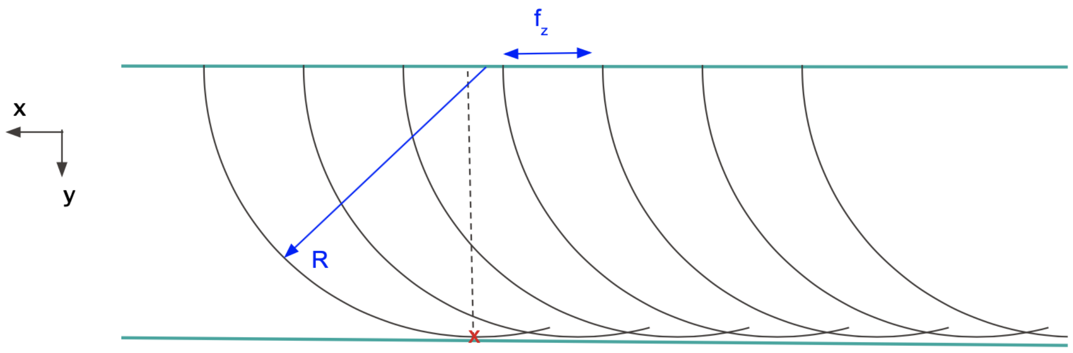


Figure 5: 2D scheme of the path followed by the tool teeth

Here, it has to be noted that an assumption was made: the path of the teeth is modeled as a circular arc but in reality this is a bit more complicated than that since the tool has a spindle speed as well as a feed rate. Hence, while being on a same circular arc, the tool is going forward at the same time in the x direction. This is however a fair approximation. Indeed, the plot of a tooth kinematic using the exact same parameters of the simulation is provided on Figure 6 and it can be stated that the trajectory is almost circular.

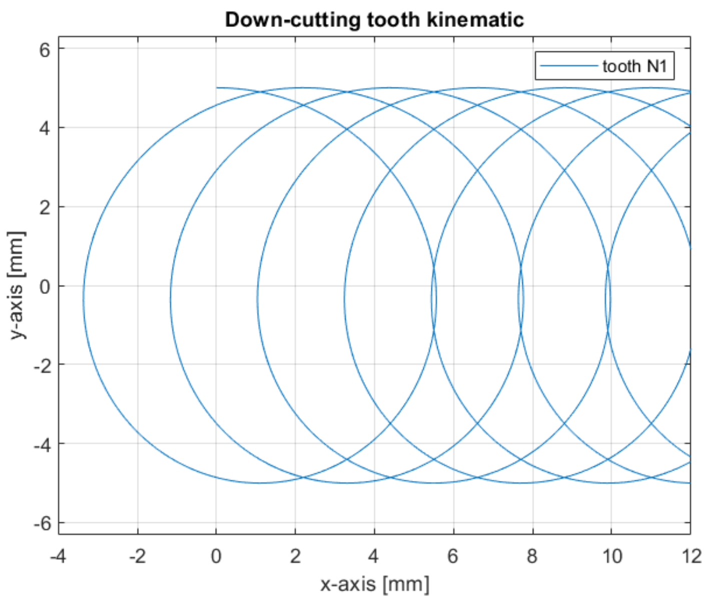


Figure 6: Down-cutting tooth kinematic for spindle speed $n = 900[rpm]$, feed rate $v_f = 2000[mm/min]$ and tool radius $R = 5[mm]$.

Now, it is possible to define the different parameters to find the number k of previous timesteps to consider.

Cutting angle ϕ_{cut}: This is the angle considered when a tooth is in direct contact with the workpiece. In the code this angle is set to $\phi_{cut} = 90°$ since the center of the tool is aligned with the workpiece surface before milling. This angle can be easily changed depending on the position of the tool.

Cutting time of one tooth Δt_{cut}
This is the time it takes for one tooth to go from the surface of the workpiece down to the surface of the already milled area. In other words this is the time needed for one tooth to travel ϕ_{cut}.

$$\Delta t_{cut} = \phi_{cut} \times \frac{60}{n}\ [s] \tag{3}$$

***Free* cutting time Δt_z**: This time is to take into consideration only if the angle between two teeth is greater than the cutting angle (i.e $360/Z_{eff} > \phi_{cut}$). It can also be described as the tool having less than four teeth. This is the theoretical time when no tooth is touching the workpiece.

$$\Delta t_z = \left(\frac{360}{Z_{eff}} - \phi_{cut}\right)\frac{60}{n}\ [s] \tag{4}$$

Number of path A to machine one final point: This has been found to be a ratio of the feed per tooth f_z and the tool radius R. In the case of a centered tool on the workpiece surface it is quite straightforward to see that for a given R, if f_z is decreased the number of paths to consider will increase (see Figure 5).

$$A = \text{roundup}\left(\frac{R}{f_z}\right) \tag{5}$$

Once again there is a slight variation if the angle between two teeth is greater than the cutting angle. In this case one unit has to be removed from A which corresponds to the *free* cutting time of the first tooth that machined the point considered.

Timestep between two measures dt: During the simulation the measures are retrieved with an interval dt $[s]$.

Finally the number of timesteps K to consider is given by Equation 6 for an angle between two teeth greater than the cutting angle.

$$K = \text{roundup}\left(A\frac{\Delta t_{cut}}{dt} + (A-1)\frac{\Delta t_z}{dt}\right) \tag{6}$$

Equivalently if the angle between two teeth is smaller than the cutting angle the Equation 7 is obtained.

$$K = \text{roundup}\left(A\frac{\Delta t_{cut}}{dt}\right) = \text{roundup}\left(A\frac{60 \cdot \phi_{cut}}{n \cdot dt}\right) \tag{7}$$

where ϕ_{cut}, when related with the radius R of the tool, parameterizes the depth of cut with the relation $a_e = R(1 - sin(90 - \phi_{cut}))$.

With the following machine parameters of the simulation:

- $f_z = \frac{v_f}{Z_{eff} \cdot n} = 0.28\ [mm]$
- $\phi_{cut} = 90°$
- $n = 900/60\ [rps]$
- $dt = 0.001\ [s]$

the Equation 7 is rewritten as:

$$K = \text{roundup}\left(A\frac{60 \cdot \phi_{cut}}{n \cdot dt}\right) = 150 \tag{8}$$

Thus, 150 previous timesteps have to be considered if all the information for a given quality measurement is to be kept.

3.6 Relating multiple timesteps to quality estimates

With the number of previous timesteps to be considered known, it remains the question of how much each of them impacts the quality at a given z-position of the tool. The first timesteps when a given z-position is being milled are maybe not as important as the last ones, where the final chips of material are being removed at that position. From another perspective, the power dissipated to cut through the workpiece is linked to the curve length traveled by the teeth of the tool. The power used at each timestep could therefore be split among all the discretized points of a workpiece being milled at that timestep.

Intuition and knowledge on the physics behind the milling operations help understand how the measured information could be related to the quality estimates. However, this is far from enough to derive an analytical function to compute this quality estimates. Here comes into play one of the strength of machine learning algorithms. Using large datasets, they should, one way or another, be able to approximate these complex relationships, as long as they are provided with enough meaningful information.

Unfortunately, by the end of the time allocated for this project, the relations found and presented until here are not enough to use the full potential of GP with the construction of kernels tailored to the milling problem (section 6.6).

4 Preprocessing

The first step in the implementation of the algorithm is the preprocessing of the data. This phase is one of the most important in machine learning because the accuracy of the results and predictions will substantially rely on the choices made during the preprocessing.

4.1 Loading

10 datasets are created for each operation (rotated workpiece and random $y - coordinate$ tracking). Each of them is first loaded independently to perform corrections and data-augmentation.

4.2 Cleaning

Only the information from timesteps where the tool is removing material from the workpiece is of interest. This means that the data related to timesteps where the spindle power is zero should be corrected.

For timesteps at which the power is zero because to tool is either not in the workpiece yet or has already milled through it, the information related should be removed without further consideration.

This is however not enough as it may happen that the power is zero while the tool is inside the workpiece. This happens when the tool is entering and leaving the workpiece or when the number of teeth is smaller than a certain value (section 3.4.2). When entering and leaving the workpiece, only a fraction of the tool is milling. There are therefore a few timesteps between two successive teeth cut through the workpiece. At these timesteps no material is removed and the power is therefore zero. Disregarding the fact that the tool is already inside the workpiece, if the number of teeth is too small to have permanently a tooth removing material, then the power may be equal to zero at some additional timesteps too. There is two approaches to deal with these cases:

Deleting approach: All the timesteps at which the spindle power is zero are simply removed.
Interpolation approach: In the *in-between* cases where the tool is inside the workpiece but no material is actually being removed, the spindle power and the power along the axis the tool is moving are corrected with the interpolated values between the closest previous and next timesteps at which the power is not zero.

4.3 Data augmentation

At each timestep of the simulation, a quality estimate is computed and associated with the spindle power and the position of the tool, the powers and the vibrations along the x, y and z-axis. This information is however not all the information related to what is the expected quality estimate along the z-axis. Following the reasoning discussed in section 3.6, the non-zero powers (among p_s, p_x, p_y and p_z) from the related timesteps derived in section 3.5 are added to the input features related to each quality estimate.
This means that the quality estimate computed at timestep $t = t_i$ is related with the input features of not only the timestep t_i, but also all the timesteps from t_{i-K}, with K the number of related timesteps. For the present case, this number is 150 (section 3.5, equation 8). Adding the non-zero powers, it increases the number of input features from 11 to 308.

4.4 Splitting

The data splitting phase defines which ratio of the initial dataset is used for the training, validation and testing phase of the algorithm. The training set will be used to train the model and optimise the parameters. The validation set serves to evaluate the model, ensuring no overfitting of the model to the training set. Eventually a testing set is used for the predictions phase and an overall score metric, in addition to the one provided by the validation set. The default ratios are presented in Table 1. More details on the use of these different sets are given in the cross-validation section 7.4.

Table 1: Data splitting ratios

Train set	**Validation set**	**Test set**
60%	30%	10%

4.5 Features selection

Correlation between input features may indicate redundant information. While this could be disregarded (section 6.1), it may be computationally expensive to keep them all, even more after having augmented the dataset with previous timesteps. To account for that, two independent steps are implemented.

First, all the constant or quasi-constant features are deleted. If the same value is present in more than 98% of the dataset, then the feature is deleted. This removes any constant feature plus eventual features where for any reason a few sparse values would be non-zero.

Then the dataset can, optionally, be inspected for correlation between the features. The metric implemented for that is the Spearman's rank correlation coefficient. Spearman correlation assesses monotonic relationships between two variables, whether they are linear or not. For each pair of features, a coefficient is computed. If it's higher than a fixed threshold, then both features are assumed too correlated. The input feature being more correlated with the others is deleted. The threshold is here a parameter to tune. The impact on the training process and the final results will be inspected.

The features that remain despite the threshold on the Spearman's rank correlation coefficient are:

Correlation threshold	# input features	Input features
0.98 to 1	307	all except constants
0.95	305	all except constants and vibrations
0.9	155	x, y, p_s, p_x, ... , $p_x(t_{i-150})$, t
0.85	12	x, y, p_s, p_x, ... , $p_x(t_{i-7})$, t
0.8	8	x, y, p_s, p_x, $p_x(t_{i-1})$, $p_x(t_{i-2})$, $p_x(t_{i-7})$, t

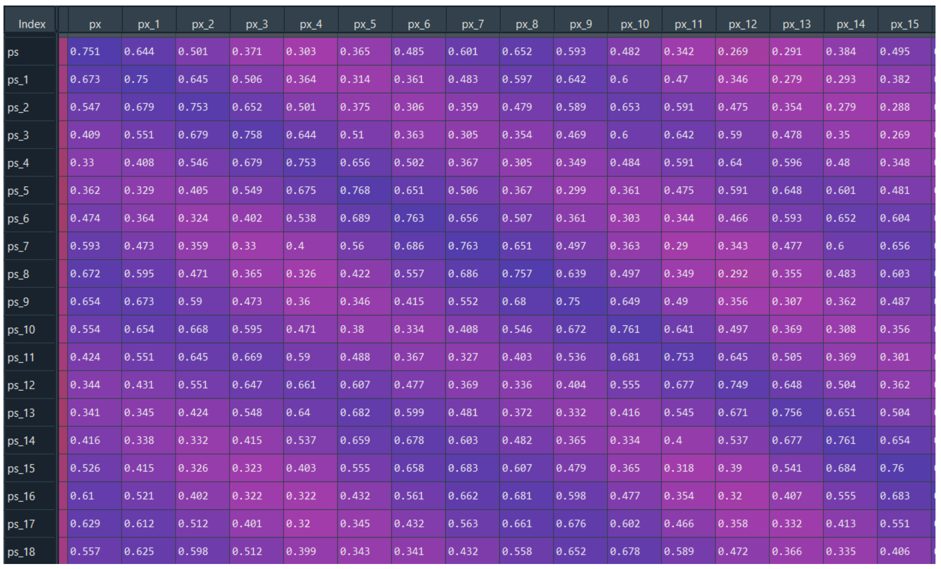
Table 2: Remaining features after selection with Spearman's rank correlation coefficient.

Some high correlation coefficients to note are between:

- p_s and p_x: $\simeq 75\%$
- p_s, vib_y and vib_z: $\simeq 97\%$
- p_x and vib_x: $\simeq 98\%$
- vib_y and vib_z: $\simeq 98\%$

The dataset is extracted from a simulation where no noise in the inputs is added. As long as the modelization of the vibrations and the power consumed are related in an monotonous fashion with the implemented formulas, the correlation between these values will be extremely high. Some correlation between p_s and p_x is also expected as the tool, for both datasets, is mainly moving along the x-axis.

It is interesting to note that the periodicity in the measurements is also observed in the correlation matrix (Figure 7). This is due to the tool geometry and rotation. The spindle speed is 900 $[rpm]$, hence a full rotation of the tool takes 66.6 $[ms]$. As the tool has 8 teeth, a tooth will enter in the workpiece every 8.3 $[ms]$, approximately every 8 timesteps.



Index	px	px_1	px_2	px_3	px_4	px_5	px_6	px_7	px_8	px_9	px_10	px_11	px_12	px_13	px_14	px_15
ps	0.751	0.644	0.501	0.371	0.303	0.365	0.485	0.601	0.652	0.593	0.482	0.342	0.269	0.291	0.384	0.495
ps_1	0.673	0.75	0.645	0.506	0.364	0.314	0.361	0.483	0.597	0.642	0.6	0.47	0.346	0.279	0.293	0.382
ps_2	0.547	0.679	0.753	0.652	0.501	0.375	0.306	0.359	0.479	0.589	0.653	0.591	0.475	0.354	0.279	0.288
ps_3	0.409	0.551	0.679	0.758	0.644	0.51	0.363	0.305	0.354	0.469	0.6	0.642	0.59	0.478	0.35	0.269
ps_4	0.33	0.408	0.546	0.679	0.753	0.656	0.502	0.367	0.305	0.349	0.484	0.591	0.64	0.596	0.48	0.348
ps_5	0.362	0.329	0.405	0.549	0.675	0.768	0.651	0.506	0.367	0.299	0.361	0.475	0.591	0.648	0.601	0.481
ps_6	0.474	0.364	0.324	0.402	0.538	0.689	0.763	0.656	0.507	0.361	0.303	0.344	0.466	0.593	0.652	0.604
ps_7	0.593	0.473	0.359	0.33	0.4	0.56	0.686	0.763	0.651	0.497	0.363	0.29	0.343	0.477	0.6	0.656
ps_8	0.672	0.595	0.471	0.365	0.326	0.422	0.557	0.686	0.757	0.639	0.497	0.349	0.292	0.355	0.483	0.603
ps_9	0.654	0.673	0.59	0.473	0.36	0.346	0.415	0.552	0.68	0.75	0.649	0.49	0.356	0.307	0.362	0.487
ps_10	0.554	0.654	0.668	0.595	0.471	0.38	0.334	0.408	0.546	0.672	0.761	0.641	0.497	0.369	0.308	0.356
ps_11	0.424	0.551	0.645	0.669	0.59	0.488	0.367	0.327	0.403	0.536	0.681	0.753	0.645	0.505	0.369	0.301
ps_12	0.344	0.431	0.551	0.647	0.661	0.607	0.477	0.369	0.336	0.404	0.555	0.677	0.749	0.648	0.504	0.362
ps_13	0.341	0.345	0.424	0.548	0.64	0.682	0.599	0.481	0.372	0.332	0.416	0.545	0.671	0.756	0.651	0.504
ps_14	0.416	0.338	0.332	0.415	0.537	0.659	0.678	0.603	0.482	0.365	0.334	0.4	0.537	0.677	0.761	0.654
ps_15	0.526	0.415	0.326	0.323	0.403	0.555	0.658	0.683	0.607	0.479	0.365	0.318	0.39	0.541	0.684	0.76
ps_16	0.61	0.521	0.402	0.322	0.322	0.432	0.561	0.662	0.681	0.598	0.477	0.354	0.32	0.407	0.555	0.683
ps_17	0.629	0.612	0.512	0.401	0.32	0.345	0.432	0.563	0.661	0.676	0.602	0.466	0.358	0.332	0.413	0.551
ps_18	0.557	0.625	0.598	0.512	0.399	0.343	0.341	0.432	0.558	0.652	0.678	0.589	0.472	0.366	0.335	0.406

Figure 7: Periodicity of 8 timesteps in the correlation matrix between p_s and p_x.

4.6 Scaling

The scaling of all datasets (training, validation and testing) is performed based on the training dataset. The validation set has the purpose of checking the validity of the trained model, it's preprocessing should be the same than the one applied to each new testing entry from the testing set. As the testing set is not known beforehand, the model can only use the dataset available for the training to estimate the distribution of the data. Assessing the validation or testing set with the dataset being scaled according to their respective mean would biased the results.

The range of values may vary between different input features. In most machine learning algorithms, the distance between two points is used in the computations. Having input features with large difference of scales may impact negatively the results, with features with high magnitude values having a large weight in the distance calculation. Scaling should nevertheless not be performed blindly. Some features may be related in such a way that their scale should remain proportional.

4.6.1 Standardization

A standardized dataset has values with zero mean and a standard deviation of 1. Rescaling the distribution of values such that the dataset becomes standardized is done by subtracting the mean of the dataset to each value and dividing by the standard deviation.

Standardization of the dataset is the rescaling most recommended for GPR. While non-necessary, it is common for GPR models to assume zero-mean output [7]. Most of the covariance functions have lengthscale parameters that will be optimised. The dataset should therefore be standardized to get a better estimation of these parameters. If the values along one dimension are much smaller than another dimension, they could be interpreted as noise. Finally, having standardize datasets can mitigate the numerical difficulties arising when inverting ill-conditioned covariance matrices.

5 Gaussian Process Regression

The objective of this section is to present the key equations behind the predictions done with exact Gaussian Process methods. For more intuition and to get an impressive visual exploration of GP, the reader is directed to the work of Jochen G., Rebecca K. and Oliver D. [8]. Additionally, most of the research done around GP is based on the fundamental work of Rasmussen and William [7]. More in-depth material on GP and detailed derivation of the equations found here below can be found on their book.

A Gaussian Process is defined as a process where any point x included in $\mathbb{R}^d$ is assigned a random variable $f(x)$ from a Gaussian distribution. GP models are non-parametric, their structure and the number of parameters are flexible and mostly determined by the data they try to fit.

A GP is therefore specified by a mean $m(X)$ and a positive definite covariance matrix $K(X, X')$. It's a distribution over functions $f(X)$, with X the function values and (X,X') any pair of points in the inputs domain:

$$f(x) \sim \mathcal{GP}(m(X), k(X, X')) \tag{9}$$

5.1 Prediction with Bayesian inference

The Bayesian inference is a method of statistical inference used in GP to compute the predictions. It derives the posterior probability $P(Y \mid X)$ (conditional distribution) from a prior probability $P(Y)$ and a likelihood function $P(X \mid Y)$ according to Bayes' theorem:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} \tag{10}$$

The joint distribution ($\mathcal{GP}$ prior) of the observed data y and the prediction y_* is given by:

$$\begin{bmatrix} y \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu \\ \mu_* \end{bmatrix}, \begin{bmatrix} K_y & K_* \\ K_*^\top & K_{**} \end{bmatrix}\right) \tag{11}$$

with $K_y = K(X, X) + \sigma_n^2 I$, $K_* = K(X, X_*)$ and $K_{**} = K(X_*, X_*)$ the covariance matrices, computed using the chosen kernel (section 6). An additive noise term, σ_n^2, is added to the covariance matrix K_y. It models the noise in the output measurements in the form of: $y = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. It's a parameter commonly add to avoid overfit and will be optimised during the training process. Note that the means μ and μ_* are set to 0 as an assumption to simplify the derivations. They are nevertheless implemented that way as the dataset is standardized before computing the matrices.

Then the conditioning rule is applied to compute the conditional distribution ($\mathcal{GP}$ posterior). Using the assumptions and the sufficient statistics on the posterior, the conditional distribution of the prediction $y_* \sim \mathcal{N}(\mu_*, \Sigma_*)$ is given by:

$$\begin{cases} \mu_* = K_*^\top K_y^{-1} y \\ \Sigma_* = K_{**} - K_*^\top K_y^{-1} K_* \end{cases} \tag{12}$$

The derivations leading to these results are not straightforward [7]. The covariance matrix K_y is often ill-conditioned, mostly due to having datapoints close to each others. This leads to quasi-identical columns in the matrix, making its inversion numerically unstable. Adding a noise term σ_y^2 to the diagonal is sometimes sufficient to successfully invert the matrix. The inversion is often implemented using the Cholesky decomposition since it's faster ($\mathcal{O}(n^3/6)$ instead of $\mathcal{O}(n^3)$) and numerically more stable. This requires the covariance matrix to be positive define, which is normally the case for GP kernels.

5.2 Limitations

Inference in GP is slow. Inverting the matrice K_y in equation (12) takes $\mathcal{O}(n^3/6)$ time. When the dataset becomes large, this becomes prohibitive. When time constraints are critical due to real-time requirements or large datasets, then the use of approximate inference schemes may help resolve the issue (section 10.2).

Gaussian processes are highly flexible. The choice among the kernels is large, hence choosing the right one is not an obvious task. When there is no prior knowledge on the function that the GP should fit, it may become a long trial-and-error procedure to choose among the vast possibilities. Automatic model construction procedures could accelerate this procedure [9].

6 Kernels or covariance functions

A kernel describes the relation between two points, x and x'. It takes them as input and outputs a similarity measure between them. Each entry K_{ij} in covariance matrix of a GP therefore describes how each pair of points in the dataset influence each other according to the kernel.

6.1 Lengthscales and automatic relevance determination

All the following kernels are parameterized by a lengthscale parameter, l. This parameter can either be a scalar (isotropic kernel) or a vector with the same number of dimensions as the input features (anisotropic kernel). The relation between the prediction and the training datapoints is set by the covariance matrix. The covariance function, called kernels, depend on the Euclidian distance between the point for which a prediction is computed and the training points used to create the GP model.

The lengthscale relates how each prediction is influenced by the training data. When the lengthscale is large, the prediction will be influenced not only with the points closer to him, but also from points farther away. On the contrary, if the lengthscale is small, then the prediction will mostly be based on the target of the training points close to him. The models with one lengthscale per input feature will factor the influence of the training points differently along each dimension.

Automatic relevance determination (ARD) refers to how the lengthscales parameters are optimised and what it implies. Through optimisation, the lengthscales should converge to values expressing the relevance of their respective input feature. A lengthscale that becomes large indicates that the function being approximated doesn't change much along that dimension, hence that this input feature isn't highly relevant in the prediction.

6.2 Scaling factor

The kernels are all multiplied by a scaling factor, σ^2, which determines the average distance of the function away from its mean. It is thus seen as an output covariance parameter.

6.3 Squared-exponential kernel

The squared-exponential kernel (SE), also called radial basis function (RBF), it's given by:

$$k(x, x') = \sigma^2 \exp\left(-\frac{d(x, x')^2}{2l^2}\right) \tag{13}$$

6.4 Rational quadratic kernel

Rational quadratic (RQ) kernel is given by:

$$k(x, x') = \sigma^2 \left(1 + \frac{1}{2\alpha} d(x, x')^\top l^{-2} d(x, x')\right)^{-\alpha} \tag{14}$$

6.5 Matérn kernel

Matérn kernel (MA) is defined as:

$$k(x, x') = \sigma^2 \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l} d(x, x')\right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{l} d(x, x')\right) \tag{15}$$

where d is the Euclidian distance between x and x', Γ is the gamma function, K_ν is the modified Bessel function of the second kind, l is the lengthscale parameter(s) and ν is the smoothness parameter.

For certain values of ν, the Matérn covariance function can be expressed as a product of an exponential and a polynomial. For example, when $\nu = \frac{5}{2}$, then:

$$k(d) = \sigma^2 \left(1 + \sqrt{5} d + \frac{5d^2}{3}\right) \exp\left(-\sqrt{5} d\right) \tag{16}$$

When $\nu = 0.5$:

$$k(d) = \sigma^2 \exp\left(-\frac{d}{l}\right) \tag{17}$$

which is equivalent to an absolute exponential kernel.

Finally, when $\lim_{\nu \to \infty}$, the Matérn kernel can be seen as a generalisation of the RBF:

$$k(d) = \sigma^2 \exp\left(-\frac{d^2}{2l^2}\right) \tag{18}$$

6.6 Adding and multiplying kernels

Kernels are highly flexible. They can be combined, by addition or multiplication, to adapt to prior knowledge one may have on the problem. The main limitation of having only a base kernel is that all the input features should be of the same type, as the regression will be based on the same kernel for all of them. The following resume is based on the work of Duvenaud [9].

Multiplying kernels is basically the AND operation for kernels: $k(x, y, x', y') = k_x(x, x') k_y(y, y')$, with x and x' two different set of values for a first list of input features and y and y' the same for a second list of input features. Two points will be considered as *close* by the similarity measure if they are close for both kernels being multiplied. Then $f(x, y)$ and $f(x', y')$ will have approximately the same value if both x and x' AND y and y' are close according to the kernels.

Adding kernels can be seen as the OR operation: $k(x, y, x', y') = k_x(x, x') + k_y(y, y')$. The covariance function will have an high value if one of the two base functions has a high value too. If x and x' OR y and y' are close, then $f(x, y)$ and $f(x', y')$ will have approximately the same value.

Adding or multiplying kernels brings the possibility of having multiple kernels, separately for each input feature.

6.6.1 Linear and periodic kernels

Common kernels to add and multiply to the ones presented previously are the linear and periodic kernels.

The linear kernel is defined as:

$$k(x, x') = \sigma_b^2 + \sigma_v^2(x - c)(x' - c) \tag{19}$$

with c the x-coordinate offset where the function has zero variance when no noise is added, σ_b^2 a constant variance and σ_v^2 the scaling factor.

The periodic kernel (sine form) is given by:

$$k(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x - x'|/p)}{\ell^2}\right) \tag{20}$$

with p the period length and l the lengthscale parameters.

Two examples, the product and the addition between a linear and a periodic kernel are shown in Figure 8. They illustrate well the difference between addition and multiplication of kernels.

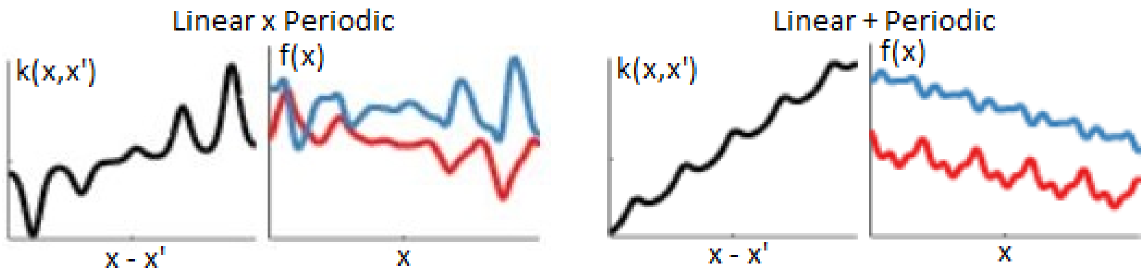


Figure 8: On the left, the covariance function $k(x, x')$ of the product between the linear and periodic kernels (in black) and two functions $f(x)$ sampled from the GP prior (in blue and red). On the right, the same for the addition of the linear and periodic kernels. Images from Duvenaud [9].

7 Optimisation of the kernel parameters

7.1 Loss function - Marginal log likelihood

To optimise the hyperparameters, the goal is to find the curve defined by the kernel function (and the related hyperparameters) that optimises the probability of generating the dataset (input features leading to output).

To quantify how well the model fits the data, the marginal log likelihood (MLL) of the GP model is computed. The marginal likelihood is given by:

$$L = p_f(y|X, \theta) = \int p(y|f(X), \theta)\, p(f(X)|X, \theta)\, df \tag{21}$$

with θ the hyperparameters of the chosen kernel, $p(y|f(X))$ the probability that the input features X would lead to the output y and $p(f(X)|X)$, the prior, probability that this set of input features X would be sampled from the inputs distribution (hence marginal).

It is often preferred to compute the logarithm of the likelihood function instead of the likelihood. The maximum of the log likelihood is reached at the same parameters than the likelihood, but by computing the log likelihood, the products between each predictions are turned into summations, which are easier to deal with. For a GP model, the prior is Gaussian ($f|X \sim N(0, K)$), hence:

$$log\, p(y|X, \theta) = -\frac{1}{2} y^T (K + \sigma_n^2 I)^{-1} - \frac{1}{2} log|K + \sigma_n^2 I| - \frac{n}{2} log 2\pi \tag{22}$$

is obtained by observing that $y \sim N(0, K + \sigma_n^2)$. The term $\frac{1}{2} y^T (K + \sigma_n^2 I)^{-1}$ is the only term related to the fit of the target. $\frac{1}{2} log|K + \sigma_n^2 I|$ expresses the complexity penalty and depends on the covariance function while $\frac{n}{2} log 2\pi$ is a normalization constant [7].

Finally, in order to optimise the hyperparameters, the gradient of the MLL is computed as:

$$\frac{\partial}{\partial \theta_j} log\, p(y|X, \theta) = \frac{1}{2} y^T K^{-1} \frac{\partial K}{\partial \theta_j} K^{-1} y - \frac{1}{2} tr\Big(K^{-1} \frac{\partial K}{\partial \theta_j}\Big) = \frac{1}{2} tr\Big((\alpha \alpha^T - K^{-1}) \frac{\partial K}{\partial \theta_j}\Big) \tag{23}$$

with $\alpha = K^{-1} y$ [7]. The optimisation problem is finally rewritten as a minimisation one to follow the convention. To maximise the MLL, the negative of the MLL derivative is therefore followed.

7.2 Optimisation solver - Stochastic gradient descent

Gradient descent algorithms are very popular when it comes to optimise the cost functions of machine learning problems. Among the most used is the stochastic gradient descent (SGD) method (Figure 9). It will be briefly presented here in the form it is implemented in this project.

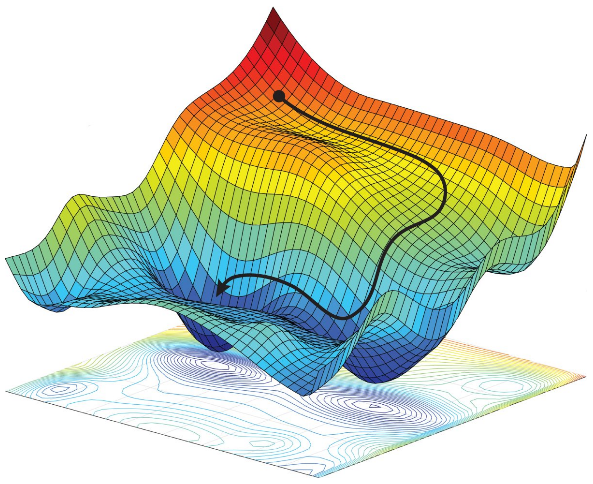


Figure 9: Visualisation of SGD path on a non-convex domain. Image from N. Azizan R. and B. Hassibi [10].

Given a new training point (x,y), the basic form of the SGD scheme updates each parameter of the model optimised with the correction term:

$$v_t = \eta \cdot \nabla_\theta J(\theta, x, y) \tag{24}$$

where $\nabla_\theta J$ is the gradient of the loss function (equation (23)) and η is the learning rate.

The parameters are updated by subtracting from the parameters the gradient term:

$$\theta_t = \theta_{t-1} - v_t \tag{25}$$

where the subscript t is for the iteration number of the optimisation loop.

Updating the parameters for each new training point makes the convergence of the algorithm noisy but also brings a great advantage, the ability to jump from a local minimum to another, potentially better. This is useful in non-convex optimisation problems, as it's the case with the quality prediction problem at hand. The drawback of this however is that there is no guarantee to converge even to a local minima, as it will always tend to overshoot.

Adding a momentum term allows to dampen this stochastic behaviour. Basically the gradient of the previous iteration is kept in memory. The term therefore increases when the gradient stays in the same direction and decreases when it fluctuates. This often leads to faster convergence with less oscillations. The momentum term added to the gradient term gives:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta, x, y) \tag{26}$$

with γ the momentum coefficient. The parameters are again updated with $\theta_t = \theta_{t-1} - v_t$.

Th SGD method can be further improved, for example with the Nesterov accelerated gradient (NAG) to give more knowledge to the momentum term. For simplicity, the scheme presented here is however the one that will be later implemented.

7.3 Score metrics

The loss function presented previously to optimise the parameters is not intuitive when it comes to analyse the results. Different metrics used later to discuss the results are presented here, along with some other common metrics found in the literature.

7.3.1 Mean absolute error

The mean absolute error (MAE) is the main metric used in the report. Its definition is intuitive, it's simply the mean absolute difference between the true target and the predicted one.

$$\text{MAE} := \frac{1}{n}\sum_{i=1}^{n} |y_{\text{i, true}} - y_{\text{i, predicted}}| \tag{27}$$

7.3.2 Mean absolute percentage error

The mean absolute percentage error (MAPE) is a normalization of the MAE metric. It's a percentage on how far the prediction is compared to the true target. It's useful to compare results between datasets with large difference in the target scales.

$$\text{MAPE} := \frac{1}{n}\sum_{i=1}^{n} \left| \frac{y_{\text{i, true}} - y_{\text{i, predicted}}}{y_{\text{i, true}}} \right| \tag{28}$$

7.3.3 Mean squared error

Mean squared error metric (MSE) is often used in the literature. It's convenient as it gives more emphasis to large errors. These are often the errors that should be corrected first and hence optimising the algorithms based on that may make sense in some situations. It's however a less intuitive metric when it comes to interpret and compare results.

$$\text{MSE} := \frac{1}{n}\sum_{i=1}^{n} |y_{\text{i, true}} - y_{\text{i, predicted}}|^2 \tag{29}$$

7.4 Cross-validation

Cross-validation is a common procedure used to measure and evaluate machine learning models. There are multiple techniques to assess how well a trained model will generalize to new data. Only K-fold cross-validation is presented here as it's the method applied during this project. This popular method is chosen because it's well adapted to limited datasets. It also results in less biased models than most other techniques as each observation in the dataset appears once in both the training set and validation set.

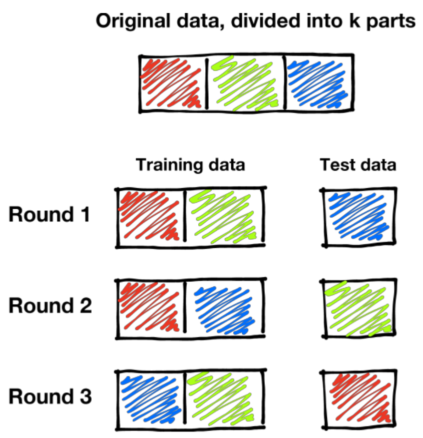


Figure 10: Overview of the K-fold cross-validation. Image from ML@B [11].

The process is quite simple as shown in Figure 10. The entire dataset is first split randomly into K folds. Then K-1 folds are used to train the model and the remaining one is used to validate it by computing the score metrics. The process is repeated until every fold as been used as the validation set. Finally, the error metrics obtained for each separate training are combined to compute a mean and a deviation value for each metric.

Usually the number of folds K is set to 10. Here however only 3 folds are used due to the high training time. While choosing larger folds leads to less biased models, the variance obtained in this work with only 3 folds is however low enough to be confident that the performances obtained are meaningful.

8 Implementation

8.1 Python and GPytorch library

The GPR approach to predict the quality estimates is implemented using Python and the GPytorch library [12]. Among all the available libraries for implementing GP, GPytorch is chosen for it's flexibility in creating complex covariance functions and its ease of use.

8.2 Algorithm

The algorithm implemented for this project follows the main outline given in this report:

1. Load the datasets and first part of preprocess (cleaning and data-augmentation).
2. Select the training and validation folds (K-fold cross-validation).
3. Second part of the preprocess (features selection and scaling).
4. Initialise the model, the kernel parameters and the likelihood function. Note that the output noise discussed in section 5, σ^2, is directly included in the initialisation of the likelihood when using the GPytorch library.
5. Run the optimisation loop. At each iteration, the model computes the MLL of the training and the validation sets and their associated losses. Using the gradient of the

training loss function, each parameter of the model is updated. The algorithm iterates until reaching convergence, when the validation loss has not decreased for 10 successive iterations (Figure 11).

6. Compute score metrics for the best model (smallest validation loss).
7. Select new training and validation folds according to the K-fold technique.
8. Repeat steps 2 to 7 until each fold as been used as validation fold.
9. Average score metrics and compute the deviation.

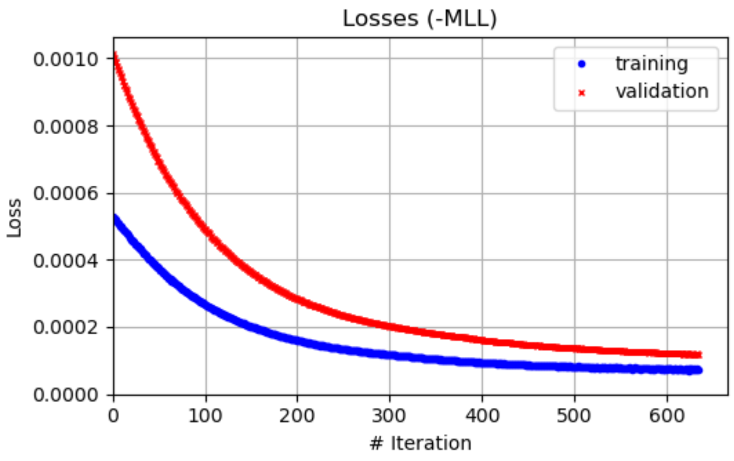


Figure 11: Losses evolution during the optimisation loop (blue: training loss, red: validation loss). Results from the training of the GP model with RQ kernel, one lengthscale per feature (section 9.1.2).

8.3 Randomness

The randomness in the algorithm comes from at least the following parts:

- Random shuffling of the dataset. Repeatability can be imposed by fixing the seed of the random generator algorithm.
- Initialisation of the parameters (lengthscales, noise). These parameters can be user defined. By defaults the GPytorch library initialises them at optimal values for the algorithm.
- Numerical instabilities when affected by them.

9 Results

The results obtained with different kernels and preprocessing choices are presented here. They are compared using the mean MAE metric from the K-fold cross-validation.

9.1 Kernels

The three most common kernels are here investigated. The preprocessing of the dataset done before fitting these models is the same for all:

- All the timesteps where no cutting is done (spindle power is zero) are deleted.
- Data-augmentation is done by including the 150 previous timesteps of power measurements into the input features.
- The correlation coefficient is set to 1, such that all features are kept for training the model.

These are also the default parameters used for comparing different preprocessing choices.

9.1.1 Matérn

Two hyperparameters are inspected for the GP with a Matérn covariance matrix: the number of lengthscales and the smoothness coefficient.

# Lengthscales	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
One per feature	13.6	1.1	249.3	8.0
One overall	14.5	6.8	293.6	2.0

Table 3: Comparison between one overall lengthscale or one lengthscale per feature, using a smoothness $\nu = 0.5$.

Smoothness ν	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
2.5	16.4	7.2	365.2	3.7
1.5	13.6	17.6	328.5	5.4
0.5	13.6	1.1	249.3	8.0

Table 4: Comparison of different smoothness, using one lengthscale per input feature.

The results indicate a better prediction when each input feature has its own lengthscale in the covariance function (Table 3 and 4). As expected, not all the input features have the same relevance.

The optimal smoothness coefficient is 0.5, pointing towards a kernel function similar to the absolute exponential. While small smoothness coefficient often leads to overfitting, the results with higher coefficients are here not better in that regard. This may reveal that the dataset is highly non-linear, hence better fitted with non-smooth functions.

9.1.2 Rational quadratic

For the RQ kernel, the only hyperparameter is the number of lengthscales.

# Lengthscales	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
One per feature	6.3	4.4	86.7	3.7
One overall	19.6	2.0	147.3	4.5

Table 5: Comparison between using one overall lengthscale or one lengthscale per feature.

As for the Matérn covariance function, one lengthscale per input feature provides better results than one overall (Table 5).

9.1.3 SE

The SE kernel also has only one hyperparameter to tune, the number of lengthscales.

# Lengthscales	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
One per feature	18.5	6.3	437.9	3.9
One overall	19.3	9.1	441.3	5.7

Table 6: Comparison between using one overall lengthscale or one lengthscale per feature.

The results are less clear than with the MA and the RQ kernel (Table 6). It's not obvious here how many lengthscales are optimal.

9.1.4 Comparison between kernels

While using one lengthscale per input feature provides better results for the MA et RQ kernels, it's less clear for the SE kernel. Between all the models tested here, the RQ covariance function gives the best results for the chosen preprocessing steps applied.

Without additional prior knowledge on the problem, no further covariance functions are tested. It should therefore be emphasized that the results shown here are just enough to compare these three kernels. No further conclusion regarding an ideal kernel for the current problem can be drawn.

9.2 Naive approaches

Kernel	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
Matérn (MA)	13.6	1.1	249.3	8.0
Squared exponential (SE)	18.5	6.3	437.9	3.9
Rational quadratic (RQ)	6.3	4.4	86.7	3.7
Naive average	1802.2	0.9	1803.2	0.5
Naive previous	15.3	52.0	15.4	104.2

Table 7: Results comparison when using the three most common kernels for prediction and two naive approaches.

To verify that the algorithms are indeed learning something useful, the results are compared with two naive approaches:

- The *naive average* approach consists in predicting always the mean value of the training set.
- The *naive previous* approach assumes that it's possible to measure the quality estimate at the previous timestep and uses this value as prediction for the actual timestep.

Compared to the *naive average* approach, all GPR models are using the data to improve the accuracy of the quality estimate (Table 7). Depending on the kernel choice with the proposed preprocessing steps, the improvement compared to the *naive average* approach ranges between a factor of 8 to 20. This is nevertheless not enough to reach the desired MAE smaller than 10 [μm].

It's obvious however that if the there was a way to measure the quality estimate at the previous timestep, as it's supposed in the *naive previous* approach, then using this value as prediction for the current timestep would be far more accurate than the proposed GPR models. Of course the reason why the control loop of the original problem is not using this naive approach is because measuring this quality estimate in real-time is not possible, hence the goal of using machine learning approaches to predict the quality.

9.3 Target shuffling

Target shuffling consists in permuting randomly the values of the target feature. The objective is to break any relationship between the input features and the output feature, here the quality estimate. After shuffling, the algorithm to be tested is trained on this *new* dataset. The accuracy of the predictions should decrease abruptly and the results obtained will set a baseline performance.

The models trained should perform better without the target feature shuffled. They should learn the relationship between the input features and the quality estimates and not just memorise the training dataset and predict based on that.

Kernel	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
MA	13.6	1.1	249.3	8.0
SE	18.5	6.3	437.9	3.9
RQ	6.3	4.4	86.7	3.7
MA shuffled	1921.9	1.0	1806.2	0.7
SE shuffled	1922.0	1.0	1824.0	0.3
RQ shuffled	1915.4	1.0	1801.2	0.3

Table 8: Results comparison when using the three most common kernels with the provided dataset and a target shuffled one.

It is interesting to note that the prediction with the target shuffling dataset have approximately the same prediction accuracy than the *naive average* approach. Therefore, based on the results (Table 8), the same conclusions as above are drawn: the models are learning meaningful knowledge on the relationship between the input features and the output.

9.4 Default preprocessing and kernel

The default parameters regarding the preprocessing steps used for comparing the kernels are also used for the subsequent tests done with different preprocessing choice. The default kernel used is the RQ kernel, with one lengthscale per input feature.

9.5 Feature selection

The objective here is to compare the results using different thresholds for the Spearman's rank correlation coefficient.

Correlation threshold	**# input features**	**# iterations**	**MAE**			
			Train		**Validation**	
			mean [µm]	rstd [%]	mean [µm]	rstd [%]
1	307	690	16.3	4.4	86.7	3.7
0.95	305	626	15.1	2.8	86.2	5.2
0.9	155	461	17.2	10.3	117.2	6.5
0.85	12	3	280.2	2.7	488.3	2.5
0.8	8	66	375.7	8.3	537.0	3.9

Table 9: Results comparison for different correlation thresholds.

The results show that keeping all the features produces the best results (Table 9). They also suggest that there is an impact on the duration of the training, requiring more iterations to converge as more parameters (lengthscales) have to be optimised. If the training time is not an issue, then it's recommended to keep them all, as the ARD automatically takes care of not taking into account the useless features (section 6.1).

9.6 Cleaning the zero power inputs

Timesteps where the tool is not cutting may not add much value to the prediction. The goal here is to get more insight on what is the effect of keeping them, deleting them or replacing the zero power values with interpolations between the previous and following timesteps where the tool is actually cutting the workpiece.

Correction method	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
None	18.1	4.7	114.8	5.8
Delete	16.3	4.4	86.7	3.7
Interpolate	15.6	13.1	85.4	3.4

Table 10: Results comparison for the correction method to apply to the zero power inputs.

The results show a clear improvement when deleting these entries from the dataset instead of keeping them as they are (Table 10). The downside is that no information on the quality can be predicted for queries with zero power as input. The model will indeed not have been trained to deal with that particular kind of inputs. Compared to the deleting approach, interpolation doesn't bring any improvements while adding data entries. It doesn't solve the problem of predicting quality estimates for queries with zero power as the model will not be trained on them as there is nothing to interpolate between when this problem arises (no *following* timestep information).

9.7 Data-augmentation

Data-augmentation	**MAE**			
	Train		**Validation**	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
Active - 200 timesteps	15.1	8.1	98.3	5.6
Active - 150 timesteps	16.3	4.4	86.7	3.7
Active - 100 timesteps	20.2	8.9	103.3	8.5
Inactive	451.7	5.3	579.1	2.4

Table 11: Results comparison with active (200, 150 and 100 timesteps) and inactive data-augmentation.

Among the different number of timesteps tested, activating the data-augmentation with the past 150 timesteps gives the best results (Table 11). It's interesting to note how without data-augmentation the prediction accuracy drops, indicating that the past timesteps bring important information for the prediction task. While more tests should be done to define exactly at how many timesteps the optimum is, the number of timesteps predicted in the problem definition (150, section 3.5) seems to approximate quite well this optimum.

9.8 Adding *Random* dataset

The previous tests have been done with all datasets from the two different operations together, without distinction. The accuracy of the predictions are here compared with the training and testing of the datasets from each operation separately.

Dataset	MAE QR			
	Train		Validation	
	mean [µm]	rstd [%]	mean [µm]	rstd [%]
Rotated x+z & Random	16.3	4.4	86.7	3.7
Rotated x+z	15.8	0.9	114.2	5.6
Random	0.036	5.1	0.306	2.9

Table 12: Results comparison with and without training the model with the *Random* dataset.

The MAE metric indicates an improvement when using both datasets to train the GP model (Table 12). However it's worth noting that the MAE metric is not normalized and therefore depend on the scale of the output being predicted. In the *Random* dataset, the true quality estimate mean is really low compared to the mean in the *Rotated* dataset (4.3 [µm] and 2789.9 [µm] respectively). This may indicate that the lower MAE using both datasets is actually due to the low error on the *Random* dataset pulling down the error from the *Rotated* dataset. Figure 12 illustrates well this difference in the quality deviations. Almost all the points that are close to the 0 value in the quality deviation axis are from the *Random* dataset, while the remaining are from the *Rotated* one.

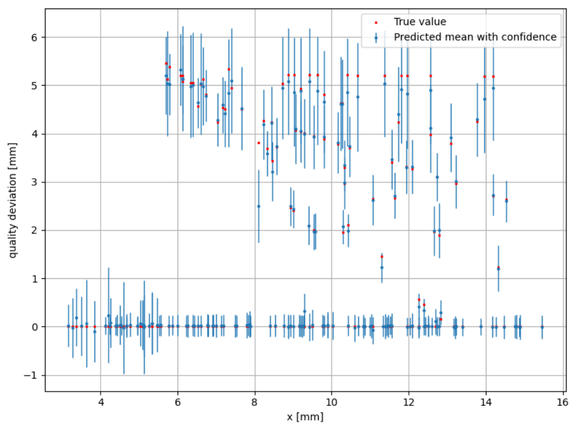
The MAPE metric is here introduced to verify these results (Table 13). It shows that indeed, when normalizing the error metric, the model performs better when trained only on the specific dataset to which it will be tested.

Dataset	MAPE QR			
	Train		Validation	
	mean [%]	rstd [%]	mean [%]	rstd [%]
Rotated x+z & Random	1.17	5.1	6.23	0.5
Rotated x+z	0.56	1.6	4.08	6.6
Random	0.84	3.8	7.01	2.5

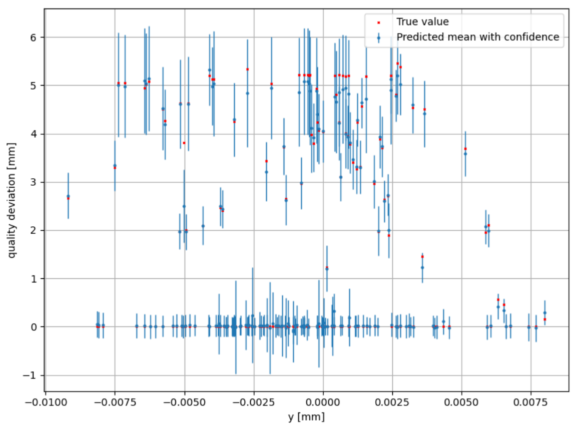
Table 13: Results comparison with and without training the model with the *Random* dataset.

These results could indicate that to optimise the accuracy of the prediction, the datasets shouldn't be mixed together. In the research done on energy prediction for milling operations (section 2.1), separating the datasets was the approach chosen. This option may be interesting when the model used for prediction will only be used for the operation it has been trained for. It may however require that the model is changed, in real-time, depending on the operation being done.

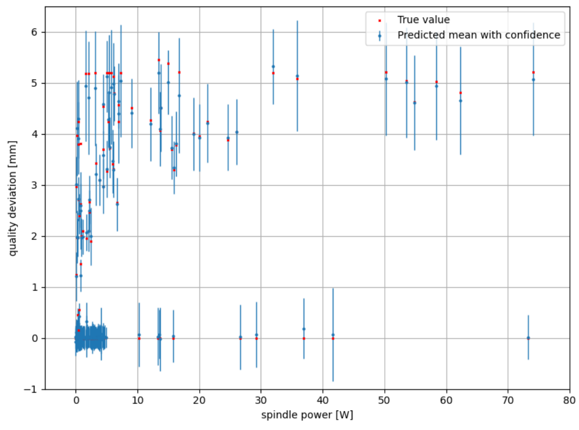
Ideally two separate datasets representing different tasks on a same machine with the same tools should be used together to improve the results in one overall model or one model for each dataset. This is known as multi-task learning, it's not further investigated here but it's an interesting direction to improve the results.



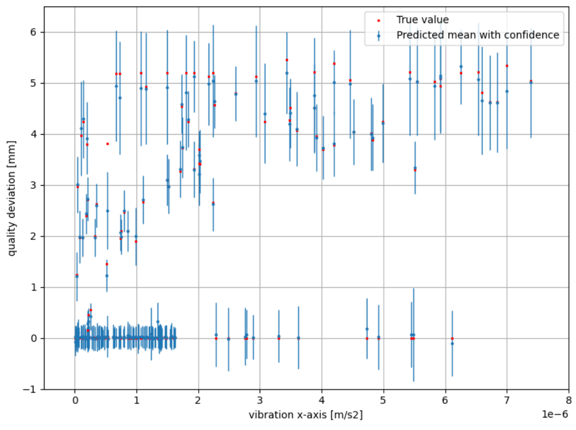
(a) x-coordinate of the tool.



(b) y-coordinate of the tool.



(c) Spindle power.



(d) Vibration on x-axis.

Figure 12: Quality deviation on a subset (random 10%) of the validation set in function of the x and y position of the tool, the spindle power and the vibration along x-axis. In red the true value of the deviation and in blue the predicted mean with the confidence interval.

10 Future prospects

10.1 Prior knowledge on the problem

Additional prior knowledge to the one already provided in the problem definition (section 3) should be acquired. It's recommended to have an in-depth mathematical analysis of the relation between the tool geometry, the power required at each timestep and the depth of cut along the meshed part. This would allow to clearly define what are the necessary parameters that the machine learning algorithm needs to approximate the function predicting the quality estimates. For the GP approach, the provided insight could help in constructing a meaningful covariance function related to the problem.

10.2 Approximate methods

The datasets used for this project are of limited size, but the computability difficulties already arise with conventional desktops. Two of the main drawbacks of GPR using exact models (section 5.2) is the storage and computational costs ($\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ respectively for a dataset with n points). Both these costs are highly prohibitive for large datasets. These limitations can however be tackled using approximation methods [7], sacrificing some accuracy for computability.

10.3 Uncertain inputs

The datasets used are extracted from simulations. They have inherently no noise. In reality, when the measurements are done with sensors, this will no longer be the case. In it's basic form, the GP model will try to fit exactly at the training points. With noisy measurements or highly non-linear relation between inputs and target, this will often lead to overfitting from the trained model. The prediction on new inputs could be improved by relaxing the fit on the training points.

11 Conclusion

Simulations are great tools for fail-safe training and validation of models. Using them in real-time to provide additional knowledge on an undergoing machining process could improve the final quality of the part manufactured.

Along this project, fundamental knowledge on the relationship between the measurements provided by the machine and the quality of the part are highlighted. It's however not enough to be able to use GP models to their full potential. This would additionally require the complete set of information from which the quality could be well approximated. More prior knowledge would also provide insight for the construction of a covariance function in adequacy with the problem at hand.

While the scoring metrics reached here are not sufficient for implementing the current algorithm, the comparative results obtained show the importance of each preprocessing step in order to improve the predictions.

Finally, it is important to emphasize that the GP methods implemented during this project are the basic building blocs of what the current state-of-the-art in Gaussian Process Regression is. The present work shows that despite not reaching the prediction objectives, GP should not be neglected for the prediction task of the quality estimate in machining operations.

References

[1] Pierre-Emmanuel Terrier. “Real-time quality control using genetic algorithm approaches”. MA thesis. EPFL, 2019.

[2] Jinkyoo Park et al. “Real-Time Energy Prediction for a Milling Machine Tool Using Sparse Gaussian Process Regression”. In: 2015. DOI: 10.1109/BigData.2015.7363906.

[3] Guojun Zhang et al. “Prediction of surface roughness in end face milling based on Gaussian process regression and cause analysis considering tool vibration”. In: *The International Journal of Advanced Manufacturing Technology* 75 (2014), pp. 1357–1370. DOI: 10.1007/s00170-014-6232-6.

[4] *Milling formulas and definition.* https://www.sandvik.coromant.com/en-gb/knowledge/machining-formulas-definitions/pages/milling.aspx.

[5] Edouard Le Goas. “Real-time quality control using genetic algorithm approaches”. MA thesis. EPFL, 2019.

[6] Francisco Montoliu Cervero. “Analysis of quality control of a milling machine using renforcement learning”. MA thesis. EPFL, 2019.

[7] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2005, pp. 27, 113–114, 171. ISBN: 026218253X.

[8] Rebecca Kehlbeck Jochen Görtler and Oliver Deussen. https://distill.pub/2019/visual-exploration-gaussian-processes/. DOI: 10.23915/distill.00017.

[9] David Duvenaud. “Automatic model construction with Gaussian processes”. PhD thesis. 2019.

[10] Navid Azizan and Babak Hassibi. “Stochastic Gradient/Mirror Descent: Minimax Optimality and Implicit Regularization”. In: *arXiv preprint arXiv:1806.00952* (2018).

[11] Machine Learning at Berkeley (ML@B). https://ml.berkeley.edu/.

[12] *GPytorch library.* https://gpytorch.ai/.